ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Desarrollo de una composición de servicios web de un proceso de negocio para una aplicación bancaria en WS-BPEL 2.0

Juan Antonio Tejero Fernández

31 de julio de 2010

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

Desarrollo de una composición de servicios web de un proceso de negocio para una aplicación bancaria en WS-BPEL 2.0

- Departamento: Lenguajes y Sistemas Informáticos
- Director del proyecto:
  - Inmaculada Medina Bulo
- Autor del proyecto: Juan Antonio Tejero Fernández

Cádiz, 31 de julio de 2010

Fdo: Juan Antonio Tejero Fernández

Agradecimientos

Quisiera agradecer en primer lugar a Inmaculada Medina Bulo, directora de este proyecto, la oportunidad de haber trabajado con ella y con un grupo de investigación formado por un gran elenco de profesores de la Escuela Superior de Ingeniería.

En segundo lugar agradecer a Antonio García Domínguez toda la ayuda prestada en la realización de este PFC, sin la cual habría resultado imposible llevarlo a cabo.

También al resto de componentes de SPI&FM que me orientaron para mejorar este proyecto.

Y por supuesto, a mi familia e Inmaculada Labrador por todo su apoyo, cariño y amor.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

1.1. Motivación.

Este proyecto arranca en un ofrecimiento por parte del grupo de investigación SPI&FM el cual pertenece al Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz. *Software Process Improvement and Formal Methods* (SPI&FM) es un grupo de investigación de la Universidad de Cádiz creado en 2004 cuya línea actual de trabajo es el estudio del lenguaje WS-BPEL 2.0.

En esta línea de trabajo se considera importante disponer de un sistema que les permita medir la calidad de distintos conjuntos de casos de prueba. Una técnica apropiada para realizar esta labor es la del análisis de mutaciones. Esta técnica utiliza una serie de operadores de mutación para generar un conjunto de mutantes a partir del programa a probar.

El objetivo principal del trabajo del grupo de investigación SPI&FM es definir un conjunto de operadores de mutación para el lenguaje WS-BPEL 2.0 que permita modelar los fallos comunes que pueden cometer los desarrolladores mientras implementan un programa en este lenguaje.

Si hablamos de WS-BPEL 2.0, SPI&FM trabaja en dos grandes herramientas que son GAmera y Takuan.

GAmera es un sistema que genera mutantes para composiciones en WS-BPEL. La técnica usada es la prueba de mutaciones, que permite mejorar la calidad de un conjunto de casos de prueba mediante la generación y ejecución de mutantes. Estos mutantes se generan mediante la aplicación de operadores de mutación al programa original. Para saber más sobre cuáles son los operadores de mutación y la función de cada uno, consultar el apéndice B.

Por su parte Takuan es una herramienta que sirve para ayudar en la prueba de composiciones de Servicios Web con WS-BPEL 2.0. Para ello Takuan usa la generación dinámica de invariantes. Esta técnica permite detectar propiedades que se mantienen

en diversos puntos del programa. Sus utilidades más inmediatas son ayudar en la depuración y verificación de una composición observando si las propiedades obtenidas están en consonancia con las especificaciones de la composición.

1.2. Objetivo del proyecto

La finalidad del PFC es en un primer instante, realizar una composición en WS-BPEL 2.0 de "gran envergadura" a la que se le puedan aplicar el mayor número de operadores de mutación posible con la finalidad de obtener un gran número de mutantes y poder trabajar con ellos. Como segunda parte del proyecto, se propuso una un poco más orientada a la investigación, en la cual se trata de conseguir un conjunto de casos de prueba de buena calidad.

La composición que debía desarrollar en un principio consistía en ampliar una ya existente denominada "LoanApproval", la cual simula la concesión de un préstamo personal por parte de alguna entidad bancaria y que había que tomar como referencia. Hay que dejar claro, que la finalidad de este PFC no era la de crear un programa que concediera préstamos personales, sino una aproximación tratando de reflejar en menor o mayor medida el procedimiento que se sigue en la vida real. El grupo necesitaba ejemplos representativos para sus técnicas y al carecer de ellos se decidió modelar un préstamo personal.

No obstante reitero que se ha tratado de hacer una aproximación bastante fiel al proceso real que tiene lugar desde que una persona visita un banco para pedir un préstamo personal hasta que se le da una respuesta. Es por eso por lo que al desarrollar esta composición se tuvo como pilar fundamental los consejos realizados por una trabajadora de un importante banco a nivel nacional experta en la concesión de préstamos.

1.3. Definiciones previas

1.3.1. Sobre SW y SPI&FM

Servicios Web: Conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web. Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas

aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

Mutantes y prueba de mutaciones: La prueba de mutaciones es una técnica de prueba basada en fallos que consiste en introducir fallos simples en el programa original mediante la aplicación de operadores de mutación. Los programas resultantes reciben el nombre de mutantes.

Puntuación de mutación: La calidad de los conjuntos de casos de prueba se mide mediante la puntuación de mutación, que indica el número de mutantes muertos frente al número de mutantes no equivalentes .

Mutante vivo y muerto: Si un caso de prueba es capaz de distinguir al programa original del mutante, es decir, la salida del mutante y la del programa original son diferentes, se dice que mata al mutante. Si por el contrario ningún caso de prueba es capaz de diferenciar al mutante del programa original, es decir, la salida del mutante y del programa original es la misma, se habla de un mutante vivo para el conjunto de casos de prueba empleado.

Mutantes equivalentes: Mutantes que presentan el mismo comportamiento que el programa original, es decir, la salida del mutante y del programa original es siempre la misma.

Operadores de mutación: Conjunto de reglas que introducen pequeños cambios sintácticos basados en los errores que suelen cometer habitualmente los programadores, o bien pretenden forzar ciertos criterios de cobertura del código.

1.3.2. Sobre conceptos relacionados con los préstamos

Préstamo: Cantidad de dinero que se solicita, generalmente a una institución financiera, con la obligación de devolverlo con un interés.

Prestatario: Es la persona titular de un préstamo. El prestatario asume todas las obligaciones y adquiere todos los derechos del contrato que firma con la entidad financiera prestamista.

Cuota: Cada uno de los pagos periódicos que se realizan para la devolución del préstamo. Se trata de la cuantía a pagar (generalmente mensuales, trimestrales o semestrales), la cual engloba el capital más los intereses.

T.A.E.: La Tasa Anual Equivalente es una referencia orientativa del coste real de una inversión o préstamo. La TAE es un indicador que, en forma de tanto por ciento anual, revela el coste o rendimiento efectivo de un producto financiero, ya que incluye el tipo de interés nominal, los gastos y comisiones bancarias y el plazo de la operación.

Aval: Se trata de un instrumento para prestar garantía del cumplimiento del pago del crédito hipotecario y sus intereses, mediante el cual una persona (avalista) se compromete a pagar las cantidades en el caso de que otra (avalado) no las hiciera efectivas.

Vida laboral: Los informes de vida laboral contienen información respecto de las situaciones de alta o baja de una persona en el conjunto de los distintos regímenes del sistema de la Seguridad Social.

CIRBE: Base de Datos en la cual vuelcan las entidades financieras la información crediticia que tiene sobre un cliente.

Registro de la propiedad: Institución que depende del Ministerio de Justicia, a través de la Dirección General de los Registros y del Notariado, y que tiene por objeto la "inscripción o anotación de los actos y contratos relativos al dominio y demás derechos reales sobre bienes inmuebles".

ASNEF: La Asociación Nacional de Establecimientos Financieros de Crédito es una asociación sindical y como tal es un enlace indispensable entre las entidades de crédito especializadas en España en financiación al consumo y las Administraciones Públicas. La ASNEF tiene un registro de morosos donde aparecen las deudas, aceptadas o no, que tenemos con entidades financieras. Es el registro más importante de España.

1.4. Antecedentes

SPI&FM tiene varias composiciones que anteceden a la desarrollada en este PFC. En torno a 10 composiciones de diversa temática son con las que han trabajado hasta el momento. Éstas, que van desde la reserva de viajes hasta la búsqueda en Google se pueden encontrar en los repositorios que posee el grupo en http://tinyurl.com/SPIFM-Samples. Si las observa detenidamente apreciará que no destacan por tener un gran

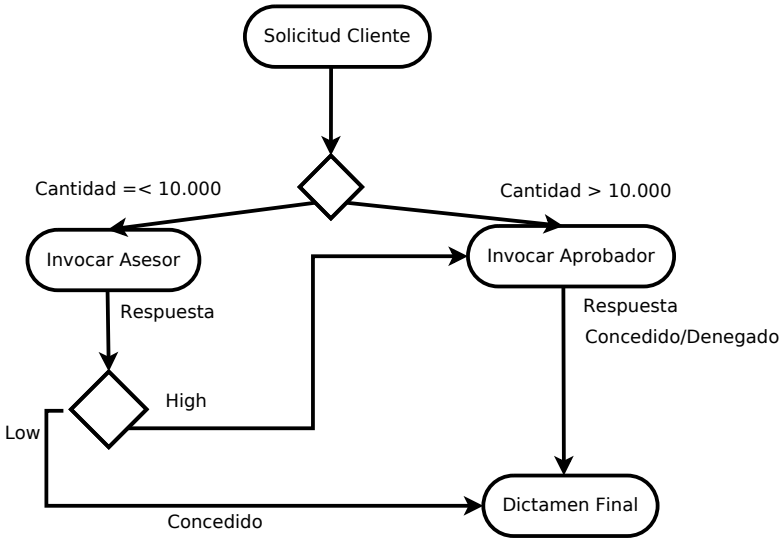

Figura 1.1: Diagrama de funcionamiento de LoanApprovalDoc

tamaño ni por poder aplicarles muchos operadores de mutación diferentes en una misma composición. De entre ellas destacamos la presencia de una que simula la concesión de préstamos personales. Ésta es LoanApprovalDoc.

LoanApprovalDoc

LoanApprovalDoc es la versión existente hasta el momento que utiliza SPI&FM para modelar una concesión de un préstamo personal. Esta composición tiene la siguiente estructura:

Un cliente solicita un préstamo de una determinada cantidad. Si la cantidad solicitada por el cliente es menor o igual a 10.000 se debe pedir ayuda a un asesor el cual nos indica el riesgo de la concesión. Si la respuesta del asesor es “alta” se debe pedir ayuda a un aprobador el cual dará el dictamen final. Si la respuesta del asesor es “baja” se concede el préstamos. En caso de que la cantidad sea mayor a 10.000, entonces invocamos al aprobador y éste nos da el dictamen final. Se aprecia de forma más sencilla observando el diagrama de la figura 1.1.

Como se puede apreciar, no es una composición excesivamente compleja. Debido a que ésta no tiene un gran tamaño y el número de operadores de mutación que utiliza son mas bien escasos se decidió ampliarla. La ampliación debía ser de tal manera que se pudiesen aplicar el mayor número de operadores de mutación y que reflejara de manera más real cómo es el proceso de concesión de un préstamo personal.

En cuanto al número de operadores de mutación que se pueden aplicar a la composición así como los resultados que se obtienen de su análisis en referencia a mutantes vivos, muertos,...se pueden observar en el apartado Pruebas y Comparativa.

1.5. GAmera

Es un sistema de generación de mutantes para composiciones WS-BPEL. Para su implementación se han utilizado 25 operadores de mutación que se clasifican en cuatro categorías atendiendo al tipo de elemento sintáctico con el que se relacionan. Las categorías se identifican mediante letras mayúsculas: I (mutación de identificadores), E (mutación de expresiones), A (mutación de actividades) y X (mutación de condiciones excepcionales y eventos). Se han definido varios operadores de mutación dentro de cada categoría. Los operadores se identifican mediante tres letras mayúsculas: la primera es el identificador de la categoría, mientras que las otras dos indican el operador dentro de la categoría. Estos operadores de mutación modelan errores comunes que pueden cometer los programadores al implementar una composición WS-BPEL. Para saber más sobre los operadores de mutación, consultar el Anexo B.

La figura 1.2 muestra los tres componentes principales del sistema: el analizador, el generador de mutantes y el sistema de ejecución.

El analizador recibe como entrada la definición del proceso original WS-BPEL y genera la información necesaria para el generador de mutantes. El generador de mutantes tiene dos componentes: un algoritmo genético en el que cada individuo representa a un mutante, y un conversor de individuos a mutantes. Este generador tiene como objetivo generar mutantes de calidad para composiciones de servicios WS-BPEL. Por último, el sistema de ejecución se encarga de ejecutar el programa original y los mutantes frente a los casos de prueba y comparar las salidas que producen.

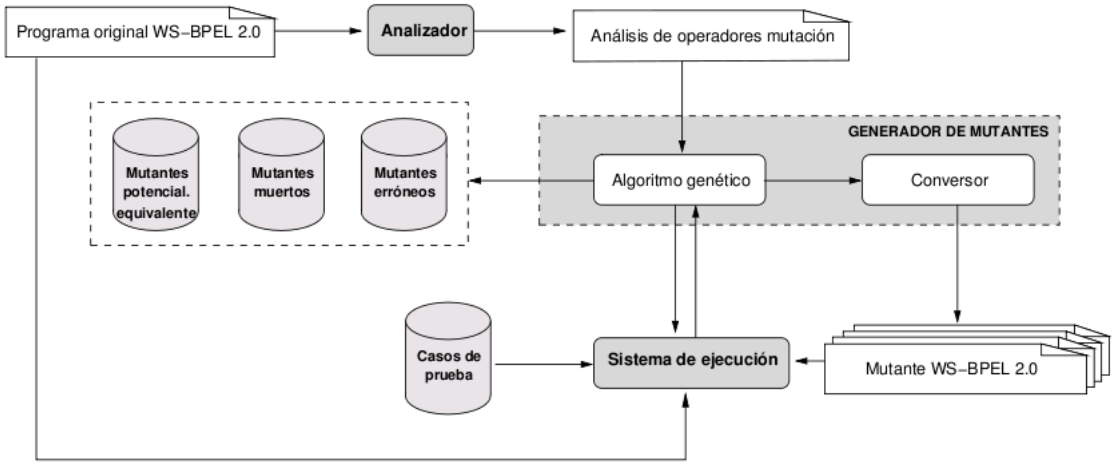

Figura 1.2: Principales componentes de GAmera

1.6. mutationtool

Una técnica apropiada para medir la calidad de conjuntos de casos de prueba es el análisis de mutaciones (*MutationAnalysis*). Esta técnica utiliza una serie de operadores de mutación para generar un conjunto de mutantes a partir del programa a probar. Con el conjunto de operadores de mutación para el lenguaje WS-BPEL 2.0 lo que se pretende es modelar los fallos comunes que pueden cometer los programadores a la hora de escribir un programa en este lenguaje .

El análisis de mutaciones es el proceso de medir la calidad de conjuntos de casos de prueba. Para ello genera un gran número de programas, denominados mutantes, que contienen una única diferencia con respecto al programa original. Los mutantes se generan aplicando al código fuente un conjunto de reglas definidas previamente, los operadores de mutación, que introducen pequeños cambios sintácticos basados en los errores que suelen cometer habitualmente los programadores, o bien pretenden forzar ciertos criterios de cobertura del código. Estos operadores introducen cambios en el programa a probar manteniendo su validez sintáctica .

Una vez generados, los mutantes se ejecutan sobre los casos de prueba; si la salida que produce el mutante es diferente de la que produce el programa original sobre un determinado caso de prueba, se dice que el mutante está muerto. En ocasiones, aparecen mutantes que siempre provocan la misma salida que el programa original, por lo que no va a existir ningún caso de prueba que permita matarlos; éstos se denominan mutantes equivalentes. Para medir la calidad de un conjunto de casos de prueba debemos eliminar los mutantes equivalentes, ya que ésta se va a calcular mediante la puntuación de mutación (*mutation score*), el cociente entre el número de mutantes

muertos y el número de mutantes no equivalentes.

En GAmera la herramienta que se encarga de realizar el *MutationAnalysis* es *mutationtool*.

2 Desarrollo del calendario

Fase de desarrollo

En esta sección del documento se detallan las tareas en las que se ha dividido el trabajo de desarrollo de este PFC. También se dispone de una representación gráfica de las misma mediante el uso de Diagrama de Gantt. Un Diagrama de Gantt es una herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. Esta representación gráfica que puede ser observada en la figura 2.1 está realizada con la herramienta GanttProject.

A continuación se describen las tareas realizadas así como el tiempo empleado en realizar las mismas. La elaboración total del proyecto ha tenido un coste en tiempo aproximado de 9 meses (desde noviembre 2009 hasta julio 2010).

Se han distinguido 8 bloques diferentes durante el desarrollo del proyecto.

- Aprendizaje de la especificación WS-BPEL 2.0.

En este primer bloque comprendido entre el 2 de noviembre y el 15 de diciembre de 2009 se estudia el lenguaje de programación que se utiliza para implementar la composición. A la labor siempre complicada de aprender un nuevo lenguaje hay que añadirle el hecho de que esta especificación está en inglés por lo que se debía estar al mismo tiempo traduciendo y adquiriendo los conocimientos del lenguaje. En este bloque también se incluye el estudio de las tecnologías WSDL, XML Schema y XPath.

- Estudio de los operadores de mutación.

Este bloque tiene un periodo de duración de unos 10 días aproximadamente (del 15 al 26 de diciembre) dedicados al estudio de los operadores de mutación, de qué se encarga cada uno y de la viabilidad de utilizarlos en la futura composición a realizar.

- Diseño general de la estructura de la composición.

Periodo comprendido entre los días 15 y 26 de diciembre, dedicado al diseño general que tendrá como estructura la composición. Se desarrolla en paralelo al estudio de los operadores de mutación ya que según se iban conociendo en mayor profundidad a los mismos, la estructura variaba para hacer uso de ellos.

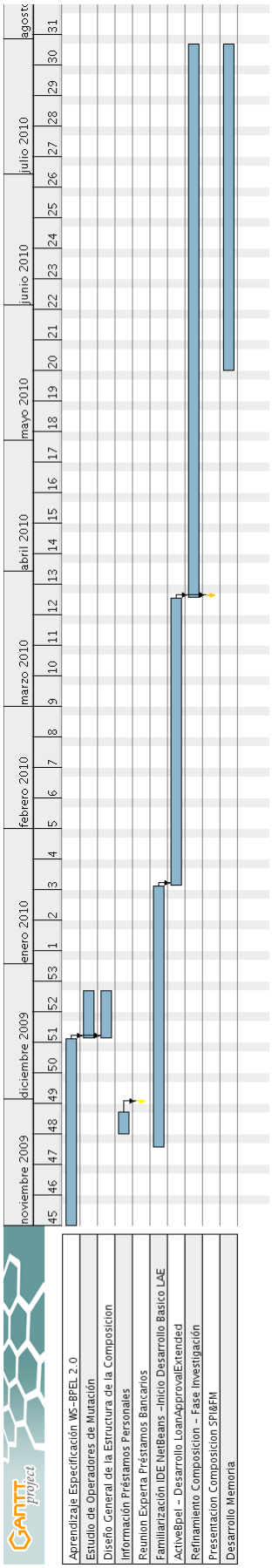

Figura 2.1: Diagrama de Gantt

- Información sobre préstamos personales.

Bloque comprendido entre el 23 y el 28 de noviembre dedicado a la recogida de información referente a los préstamos personales y todo el proceso que se debe seguir en las concesiones de los mismos.

- Familiarización con el IDE Netbeans - Inicio del desarrollo básico de LoanApprovalExtended.

Este periodo tiene una duración de un par de meses aproximadamente, desde el 20 del noviembre hasta el 19 de enero de 2010 y se dedica a la familiarización con la herramienta que nos puede servir como punto de partida para crear un proyecto en WS-BPEL. Con NetBeans se creó la estructura principal que tiene la composición.

- Desarrollo de LoanApprovalExtended.

Bloque comprendido entre el 19 de enero y el 26 de marzo de 2010 dedicado a la implementación de la composición. Aunque el núcleo principal de la composición se cerró el 26 de marzo se han ido realizando pequeños retoques hasta las últimas fechas corrigiendo errores que habían en el código detectados con la ayuda de GAmera.

- Refinamiento de la composición. Fase investigación.

Este bloque engloba a la segunda fase del PFC. En él se trata de obtener un conjunto de casos de prueba de calidad. Durante este proceso se pueden detectar errores en la composición lo cual contribuye a un refinamiento de la misma.

- Desarrollo de la memoria.

Este bloque comenzó una vez finalizada casi por completo la fase de implementación de código y al poco de comenzar la etapa de eliminación de mutantes vivos (Fase de investigación). En él se ha realizado el presente documento y ha tenido una duración aproximada de 2 meses y medio.

Además en el diagrama encontramos 2 reuniones sumamente importantes para el desarrollo de la composición: una, con la experta en préstamos personales y otra, con los miembros de SPI&FM en la que se dio por cerrada la fase de desarrollo de la composición.

3 Descripción general del proyecto

3.1. Perspectivas del proyecto

LoanApprovalExtended es una composición que modela la concesión de un préstamo personal a un cliente y aunque en gran medida refleja los pasos reales que se tienen en cuenta en dichas concesiones, no podemos "venderla" como un programa de concesión de prestamos personales. Ésta es una composición orientada a la investigación ya que su finalidad es probarla en GAmera y el objetivo era proporcionar al grupo de investigación SPI&FM una composición de un gran tamaño y a la que se le pudiera aplicar el mayor número de operadores de mutación posible.

Cuando realizamos una composición en WS-BPEL 2.0 debemos tener en cuenta que con la composición (fichero con extensión *.bpel*) propiamente dicha no basta. Existen otros lenguajes y tecnologías que se deben manejar con destreza para crear una composición completa. Con XPath 1.0 podemos realizar consultas y escribir expresiones dentro del propio código WS-BPEL. Otros lenguajes también forman parte de la composición de manera externa como es el caso de WSDL (archivo *.wsdl*) el cual nos permite describir la interfaz entre la composición y los servicios web. Además para definir nuestros tipos de datos hacemos uso de XML Schema (fichero *.xsd*). En ocasiones estas definiciones se pueden incluir dentro del .wsdl pero de manera externa permite una estructuración más clara.

Con BPELUnit tenemos un marco donde ejecutar nuestras composiciones sin necesidad de conectarlas realmente con el mundo exterior. Los Servicios Web reales son sustituidos por *mockups*. Éstos son ficheros en extensión *.bpts* en los que podemos definir mensajes que entrarán en la composición así como los que se esperan que salgan de la misma.

En la figura 3.1 observamos cómo interaccionan entre sí.

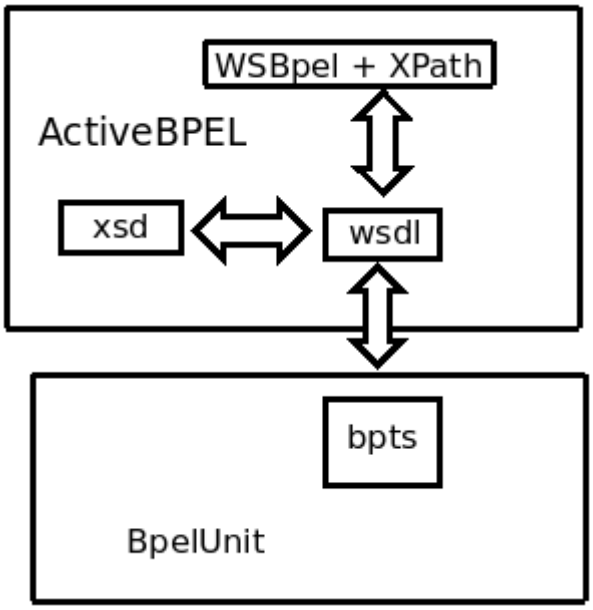

Figura 3.1: Interacción de lenguajes

3.2. Conceptos básicos de los lenguajes incorporados en la composición

3.2.1. Lenguaje WS-BPEL 2.0

WS-BPEL es un lenguaje basado en XML que permite especificar el comportamiento de un proceso de negocio basado en interacciones con servicios Web (SW). Nació como combinación de WSFL (Web Service Flow Languaje de IBM, orientado a grafos y basado en el control de los links entre tareas) y XLANG (Web Services for Business Process Design de Microsoft, basado en un control de flujos con secuencias, condiciones y bucles, etc...) y ha evolucionado adquiriendo lo mejor de cada uno e intentando evitar las malas prácticas de los mismos.

La estructura de un proceso WS-BPEL se divide en cuatro secciones:

- Definición de relaciones con los socios externos, que son el cliente que utiliza el proceso de negocio y los Servicios Web a los que llama el proceso.
- Definición de las variables que emplea el proceso.
- Definición de los distintos tipos de manejadores que puede utilizar el proceso. Pueden definirse manejadores de fallos, que indican las acciones a realizar en caso de producirse un fallo interno o en un SW al que se llama. También se definen los manejadores de eventos, que especifican las acciones a realizar en caso de que el proceso reciba una petición durante su flujo normal de ejecución.

- Descripción del comportamiento del proceso de negocio; esto se logra a través de las actividades que proporciona el lenguaje.

3.2.2. Lenguaje WSDL

Web Services Description Language 2.0 es un formato XML para describir servicios de red como un conjunto de *endpoints* (punto de entrada a un servicio o procedimiento) que operan sobre mensajes con información orientada-a-documentos u orientada-a-procedimientos.

WSDL describe la interfaz pública a los servicios Web. Describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

3.2.3. XML Schema

Es un lenguaje utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. Con él podremos definir nuestros propios tipos de datos que usaremos posteriormente en la composición.

3.2.4. XPath

XML Path Language permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos. Usando XPath crearemos expresiones por ejemplo, para trabajar con las diferentes partes que componen un mensaje o para realizar operaciones entre ellas.

Hasta aquí los lenguajes que forman parte de la composición. A continuación se describe el lenguaje utilizado para realizar este documento.

3.3. Otros lenguajes utilizados

Lenguaje LaTeX

Es un lenguaje de marcado para documentos, y un sistema de preparación de documentos, formado por un gran conjunto de macros de TeX, escritas inicialmente por Leslie Lamport (LamportTeX) en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es software libre bajo licencia LPPL.

LaTeX es un procesador de textos que está formado mayoritariamente por órdenes (macros) construidas a partir de comandos de TeX lenguaje «de bajo nivel», en el sentido de que sus acciones últimas son muy elementales pero con la ventaja añadida, en palabras de Lamport, de «poder aumentar las capacidades de LaTeX utilizando comandos propios del TeX descritos en The TeXbook» . Esto es lo que convierte a LaTeX en una herramienta práctica y útil pues, a su facilidad de uso, se une toda la potencia de TeX. Estas características hicieron que LaTeX se extendiese rápidamente entre un amplio sector científico y técnico, hasta el punto de convertirse en uso obligado en comunicaciones y congresos, y requerido por determinadas revistas a la hora de entregar artículos académicos.

3.4. Fases del proyecto

3.4.1. Funciones del proyecto

Modelado de concesión de préstamo personal. LoanApprovalExtended.

El enfoque que se le ha dado a esta composición es la de un trabajador de una entidad bancaria el cual tiene que responder a un cliente sobre la aprobación o denegación de un préstamo. El cliente desea saber si le concederá el préstamo y las condiciones de reembolso del mismo, por lo cual se le debe contestar con un sí y la cuota mensual que deberá pagar durante el periodo de reembolso o el no, con la razón de la denegación.

Cuando se realiza el estudio de una concesión en situaciones reales se hace uso de un sistema informático llamado *SCORING*, el cual ayuda al trabajador del banco proporcionando información imprescindible a tener en cuenta. No obstante la última palabra sobre la concesión la tiene el propio trabajador ya que existen factores que una máquina no es capaz de tener en cuenta.

Centrándonos en la lógica que se debe seguir para la concesión de un préstamo personal, a continuación se detallan algunos de los aspectos más importantes que se deben tener en cuenta:

- **Estabilidad personal**. ¿Probabilidades de que desaparezca el cliente?. Una de las consideraciones que se tiene en cuenta al conceder un préstamo es la de la posibilidad de desaparición por parte del cliente una vez que se ha efectuado el abono del préstamo. Por ello es importante saber por ejemplo los compromisos familiares.
- **Atributos**. Considerar las habilidades y conocimientos del cliente. ¿Es probable que mantenga un trabajo estable?. ¿Se podría a volver a colocar con facilidad?. Para estos puntos es muy importante conocer su vida laboral.
- **Pago del préstamo y garantía**. Hay que informarse de la viabilidad del reembolso del préstamo y de si hay suficiente cobertura. Para este punto es sumamente importante el estudiar la posibilidad de exigir garantías de reembolso, como puede ser la figura de un avalista.
- **Obligaciones**. Si posee hipoteca o préstamos, deseamos saber la cuantía mensual que desembolsa en ellas.
- **Propiedades del prestatario**. Las propiedades que posee un prestatario son muy importante ya que en caso de no abonar el reembolso, se le puede embargar alguna de ellas. Por tanto tenemos que valorar de manera positiva el hecho de que el cliente tenga alguna propiedad como puede ser un coche, casa, terreno.

- **Morosidad**. Es necesario saber si nuestro cliente es moroso o no.

3.4.2. Características de los usuarios

Como se comentó anteriormente éste es un proyecto orientado a la investigación, luego el perfil de usuario que necesita trabajar con él será casi en exclusividad los propios miembros del grupo SPI&FM que lo deberán ejecutar bajo GAmera y Takuan.

Los usuarios se podrán encontrar con una composición a la que se le pueden aplicar 18 de los 25 operadores de mutación que de momento se han considerado oportunos por el propio grupo, con un total de 40 casos de pruebas creados y con un total de 3374 mutantes generados.

Aunque por sí misma la ejecución de la composición no tiene complejidad alguna puesto que no hay ningún tipo de interactividad, el análisis posterior de los resultados de los resultado sí que podría suponer algún problema para un usuario inexperto.

Otro perfil de usuario sería el de un estudiante que desease realizar otro PFC similar o alguna composición en WS-BPEL y que ésta le sirviera como simple referencia para iniciarse en estas tecnologías.

3.4.3. Suposiciones y dependencias

Para ejecutar LoanApprovalExtended necesitamos tener instalada GAmera en nuestro sistema ya que haremos uso de ella (o parte de ella mejor dicho). Si atendemos a las dependencias de GAmera, ésta tiene que ejecutarse bajo un entorno Sun Java 6. De forma transitiva LoanApprovalExtended depende de:

- **Tomcat 5.5** Tomcat funciona como un contenedor de *servlets*. Tomcat implementa las especificaciones de los *servlets* y de JavaServer Pages (JSP) de Sun Microsystems. Tomcat es un servidor web con soporte de *servlets* y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en *servlets*. El motor de *servlets* de Tomcat a menudo se presenta en combinación con el servidor web Apache.
- **ActiveBPEL 4.1** Es una implementación de código abierto de un motor para *BPEL* escrito en *Java*. Se ejecuta en cualquier contenedor *servlet* estándar.

- **BPELUnit** Framework de pruebas para WS-BPEL.

Aunque no es una dependencia directa de la composición, sería conveniente tener *Python* instalado ya que se hace uso de un guión en dicho lenguaje para realizar una ejecución completa y exhaustiva de la composición. Con este guión conseguimos un fichero de salida en el que podremos observar todos los datos relacionados con tiempos de ejecución, mutantes muertos y qué casos de pruebas matan a qué mutantes.

Del mismo modo se recomienda usar *TkDiff*. Con él conseguimos ver con un simple golpe de vista dónde radican las diferencias entre dos ficheros. Es sumamente útil cuando estamos en la fase de matar mutantes y deseamos saber en qué parte del código original se ha producido la mutación.

4 Desarrollo del proyecto

4.1. Desarrollo del proyecto

4.1.1. Descripción del funcionamiento de LoanApprovalExtended

A continuación se describe a grandes rasgos el funcionamiento de LoanApprovalExtended: La composición recibe como entrada varios datos entre los que destaca la cantidad que solicita el cliente así como el número de meses en los que se desea reembolsar la cantidad (a la que se le aplicará un interés de beneficio por parte de la entidad). Además se reciben otros datos que servirán a la composición para determinar la viabilidad de la concesión. Éstos corresponden a gastos mensuales que puede tener el cliente así como otros datos personales como son su nivel de estudios, estado civil, etc.

Hay que mencionar que se han distinguidos 5 grupos de profesión, 5 estados civiles y 6 niveles de estudios, los cuales se pueden observar en los cuadros 4.1 , 4.2, 4.3 respectivamente.

Es en este punto donde encontramos el primer filtro que aplica la composición. La cantidad solicitada no debe ser menor a 3.000 €, ni tampoco mayor a 60.000 €. Si se sobrepasan estos límites se deniega el préstamo y la composición devuelve un mensaje comunicándolo.

Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Abogado	Creativo	Sector fotográfico	Jubilado	Ama de casa
Arquitecto	Mando intermedio	Comercial		Desempleado
Directivo	Programador	Comerciante		Estudiante
Gobierno	Secretario	Dependiente		
Funcionario				
Ingeniero				
Médico				
Profesor				

Cuadro 4.1: Cuadro de grupo de profesión

Soltero
Viudo
Separado
Divorciado
Casado

Cuadro 4.2: Cuadro de estado civil

Primarios incompletos
Primarios completos
E.S.O.
Bachillerato
Diplomatura/Ingeniería Técnica
Licenciatura/Ingeniería

Cuadro 4.3: Cuadro de nivel de estudio

El segundo filtro también usa los datos de entrada. En él se calcula un cociente entre los meses en los que se desea reembolsar el préstamo y la cantidad solicitada. Éste cociente no debe ser ser superior a 0.0144 (se estableció este valor en base a recomendaciones de una profesional del sector).

Una vez pasado el anterior escollo, se ejecutan al mismo tiempo las llamadas a los cuatro servicios Web que nos aportarían la información restante del cliente. Estos cuatro servicios web contemplados son los siguientes:

- Seguridad Social
- Banco de España
- Ministerio de Justicia
- ASNEF

Hay que avisar al lector que estos cuatro servicios realmente no existen, sino que han sido inventados para realizar la composición y del mismo modo ocurre con los datos (y el formato de éstos) que a través de los servicios web recibe la composición.

Seguridad Social

Este servicio Web nos proporciona un informe de la vida laboral del cliente. De ella podremos obtener datos sumamente importantes como es la situación laboral actual. Si el cliente no trabaja en la actualidad se le denegará el préstamo. En caso de que sí trabaje, hay información de su historial laboral que hay que tener en cuenta: el número de días transcurridos desde su primer trabajo, el número de días trabajados desde su

primer trabajo y el número de trabajos a lo largo de su vida. Estos datos pueden ser utilizados para buscar al cliente que cumpla el perfil ideal para una entidad prestamista, "mucho trabajado en pocos trabajos". Este perfil deseado responde a la necesidad que tienen las entidades de buscar una persona estable que les garantice que no perderá el trabajo actual mientras devuelve el préstamo y en caso de perderlo, que sea capaz de incorporarse con rapidez de nuevo al mundo laboral.

Banco de España

Con este servicio Web obtendríamos los datos que posee la Cirbe (Base de Datos del Banco de España con información crediticia) sobre el cliente. La información proporcionada consiste en un listado de los préstamos e hipotecas que posee el cliente en la actualidad, en caso de tenerlos.

Ministerio de Justicia

Del Ministerio de Justicia obtendríamos el Registro de la Propiedad a través del cual conoceríamos si el cliente posee alguna propiedad a su nombre y el estado de la misma. Es importante esta información ya que en caso de ser necesario, la entidad prestamista podría embargar alguna propiedad poseída por el prestatario. Si el cliente posee ya alguna propiedad embargada, se le denegará el préstamo.

ASNEF

La Asociación Nacional de Establecimientos Financieros de Crédito tiene un registro sobre todos los morosos que existen en España. Este servicio Web proporciona la información de morosidad referente al cliente solicitante. Si el cliente es moroso, se le denegará el préstamo.

Si la ejecución de la composición sigue su curso satisfactorio tras evaluar individualmente los datos obtenidos en cada servicio Web, a continuación se valúan los datos obtenidos de la entrada referentes a gastos mensuales junto con los datos obtenidos del Banco de España para llevar a cabo el cálculo de lo que se conoce como el Canon Francés. Éste dice que los gastos mensuales que debe tener una persona no debe sobrepasar el 40 % de los ingresos mensuales que tiene el individuo. Si los sobrepasa, se le denegará el préstamo.

Por otro lado, se puede considerar necesaria la figura de un avalista que avale la operación. Éste será necesario en las siguientes situaciones:

- El cliente trabaja actualmente pero su contrato es temporal.

- El cliente trabaja actualmente pero su vida laboral tiene poca duración. Se ha establecido un periodo mínimo de días trabajados en la vida laboral que deberá sobrepasar el cliente. Este periodo son 180 días. Si a lo largo de su vida el cliente ha trabajado menos de 180 días deberá aportar un aval.

Al aval también se le evaluará y no debe ser ni moroso ni tener ningún embargo a su nombre.

Por último se realiza una evaluación global de los datos obtenidos hasta el momento, tanto recibidos del cliente como a través de los servicios Web. Esta evaluación deberá pasar un umbral. Si no se pasa se deniega el préstamo. Si se pasa se le concederá al cliente el préstamo.

En la figura 4.1 se muestra gráficamente el funcionamiento de la composición descrito anteriormente.

4.1.2. Análisis del sistema

Para el desarrollo del proyecto se debe seguir una metodología de desarrollo, la cual se puede considerar como una serie de pasos y procedimientos que deben seguirse para desarrollar un producto software.

La metodología seguida ha sido Métrica Versión 3 Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información, propuesta por el Ministerio de Administraciones Públicas, ya que de momento ha sido la única con la que se ha trabajado en la Ingeniería Técnica.

MÉTRICA versión 3 puede ser utilizada libremente con la única restricción de citar la fuente de su propiedad intelectual, es decir, el Ministerio de Administraciones Públicas.

METRICA Versión 3 ha sido concebida para abarcar el desarrollo completo de Sistemas de Información sea cual sea su complejidad y magnitud, por lo tanto su estructura responde a desarrollos máximos y debe adaptarse y dimensionarse en cada momento de acuerdo a las características particulares de cada proyecto. En este capítulo se incluyen los modelos considerados más relevantes del proceso de Desarrollo de Sistemas de Información, que incluye los siguientes subprocesos:

- Estudio de Viabilidad del Sistema (EVI)
- Análisis del Sistema de Información (ASI)
- Diseño del Sistema de Información (DSI)
- Construcción del Sistema de Información (CSI)
- Implantación y Aceptación del Sistema (IAS)

Estudio de la viabilidad del sistema

Establecimiento del alcance del sistema

El propósito de este proceso es analizar un conjunto concreto de necesidades, con la idea de proponer una solución a corto plazo. Los criterios con los que se hace esta propuesta no serán estratégicos si no tácticos y relacionados con aspectos económicos, técnicos, legales y operativos.

Debemos estudiar el alcance de la necesidad planteada por el cliente o usuario; en el caso de LoanApprovalExtended, el Proyecto fue planteado por el propio grupo de investigación de la Universidad de Cádiz SPI&FM. LoanApprovalExtended es una

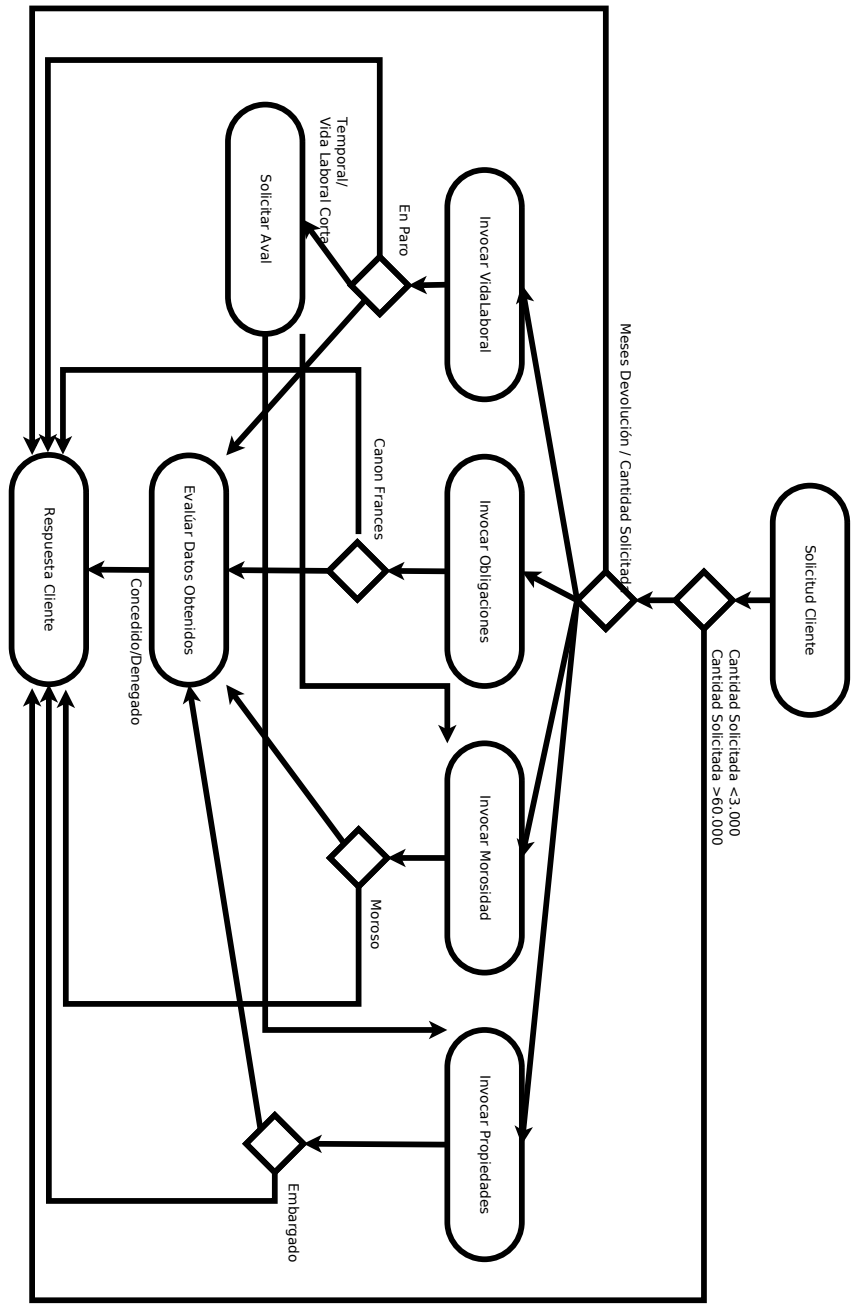

Figura 4.1: Diagrama de funcionamiento de LoanApprovalExtended

composición necesaria porque actualmente el grupo de investigación no posee composiciones de gran tamaño y que además abarque al mayor número posible de operadores de mutación.

Estudio de la situación actual y predefinición de requisitos

Actualmente las composiciones con las que trabaja el grupo tienen en su mayoría una media de unos 50 ó 60 operandos de mutación por composición y según comentan ellos mismos: “Hemos probado nuestro sistema con varias composiciones WS-BPEL de ejemplo, obteniendo buenos resultados. Por ello nuestro siguiente paso es probarlo con composiciones más complejas para evaluar más a fondo su valía y poder mejorarlo de cara a su adopción como herramienta de apoyo a la prueba de WS-BPEL ”.

Como requisitos además de tener instaladas las aplicaciones mencionadas, lo más importante es tener los conocimientos necesarios para poder interpretar los resultados obtenidos y entender la necesidad de la existencia de software de pruebas de código y por extensión de composiciones de gran tamaño que ayuden a evaluar la valía de ese software de prueba.

Estudio y valoración de alternativas de solución

El nombre LoanApprovalExtended responde a que ésta en un principio iba a ser una composición que extendiese a una ya existente creada por el propio grupo (*LoanApproval*), pero teniendo en cuenta que esa composición no reflejaba el proceso real de la concesión de un préstamo se acabó decidiendo el realizar una desde cero. No obstante se mantuvo el nombre de extensión de la que ya había.

En cuanto al lenguaje de programación no existen alternativas posibles ya que desde un principio el objetivo era trabajar con WS-BPEL y no con ningún otro lenguaje de programación.

4.2. Análisis del sistema

El propósito de este proceso es conseguir la especificación detallada del sistema de información a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

Para el desarrollo de un proyecto es muy importante realizar un modelado tanto de análisis y diseño para obtener un conjunto de modelos que nos permitan especificar los requisitos, la estructura y el comportamiento del sistema.

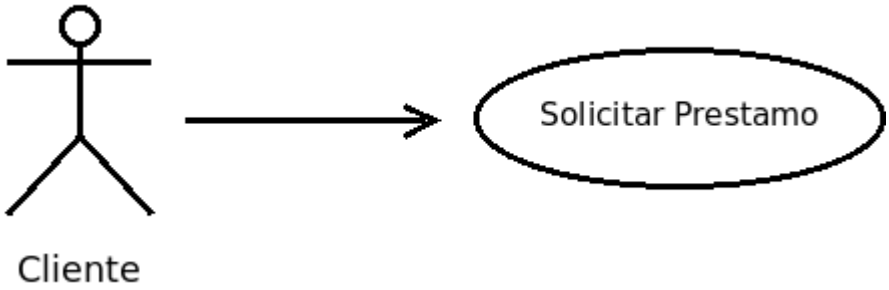

Figura 4.2: Diagrama de Casos de Uso

Comenzaremos nuestro análisis del sistema realizando el modelado de casos de uso.

4.2.1. Modelado de casos de uso

Describiremos los casos de uso de nuestro proyecto ayudándonos de representaciones gráficas, diagramas de casos de uso y de una descripción del mismo.

Los casos de uso son una técnica de especificación de requisitos válida tanto en desarrollos estructurados como en orientación a objetos, aunque en este último caso se propone como técnica obligatoria al ser necesaria como referencia a lo largo de todo el ciclo de vida.
En esta tarea se elabora el modelo de casos de uso, según las normas y estándares de la organización, identificando:

- **Actores**: Un actor representa un conjunto coherente de roles que juegan los usuarios de los casos de uso al interaccionar con el sistema .
- **Casos de uso**:Un caso de uso especifica el comportamiento deseado del sistema. Representan los requisitos funcionales del sistema. Describen qué hace el sistema, no cómo lo hace .
- **Breve descripción de cada caso de uso**: En la que entre otros, identificaremos al escenario principal (Interacciones entre los actores y el sistema necesarias para obtener el objetivo) y los posibles escenarios alternativos(flujos de ejecución en los que llegamos a estado de error u otras posibles soluciones al caso de uso)

Diagrama de casos de uso

Un diagrama de Casos de Uso es una representación gráfica en la que podemos observar visualmente qué hace nuestro sistema. En la figura 4.2 vemos el diagrama de casos de uso de la composición.

Especificación del caso de uso

El objetivo de esta tarea es especificar cada caso de uso identificado en la tarea anterior, desarrollando el escenario. Para completar los casos de uso, es preciso especificar información relativa a:

- **Descripción del escenario**, es decir, cómo un actor interactúa con el sistema, y cual es la respuesta obtenida.
- **Precondiciones y poscondiciones**.
- **Identificación de interfaces de usuario**.
- **Condiciones de fallo** que afectan al escenario, así como la respuesta del sistema (escenarios secundarios).

Caso de uso: Solicitar préstamo personal

Actor Principal: Cliente de la entidad que quiere solicitar un préstamo.
Personal involucrado e intereses:
Cliente: Desea que le concedan un préstamo bancario.
Trabajador del banco: Desea que los clientes acudan con todos sus datos correctos.
Entidad bancaria: Desea que el mayor número de clientes posibles la elijan como entidad para solicitar préstamos personales.
Precondición: El cliente proporciona correctamente todos sus datos.
Postcondición: Al cliente se le concede el préstamo.

Escenario principal de éxito (o de Flujo Básico):

1. El cliente llega a la entidad para solicitar el préstamo.
2. El trabajador de la entidad comienza una nueva solicitud de préstamo introduciendo el DNI del cliente así como la cuota mensual que paga de seguro de vida, de hogar, agua, luz y su salario.
3. El sistema almacena la información.
4. El trabajador introduce la cantidad solicitada, el número de meses del reembolso, el grupo de profesión al que pertenece, el estado civil y el nivel de estudios del cliente.
5. El sistema los almacena y comprueba que sean correctos.
6. El sistema conecta con los diferentes servicios web.
7. El sistema comprueba que los datos obtenidos de los servicios web son correctos.
8. El sistema da como válidos los resultados obtenidos tras analizar los datos.
9. El sistema considera que no es necesario ningún aval.

10. El sistema determina que se concede el préstamo.
11. El trabajador comunica al cliente el dictamen.
12. El cliente se marcha con su préstamo concedido.

Extensiones (O Flujos Alternativos):

5a. El sistema comprueba que hay un error en el nivel de estudios introducido:
    1. El sistema indica el error y finaliza la ejecución.

5b. El sistema comprueba que hay un error en el grupo de la profesión introducida:
    1. El sistema indica el error y finaliza la ejecución.

5c. El sistema comprueba que hay un error en el estado civil introducido:
    1. El sistema indica el error y finaliza la ejecución.

5d. El sistema comprueba que la cantidad solicitada no está en el rango permitido:
    1. El sistema indica el error y finaliza la ejecución.

5e. El sistema comprueba que el número de meses de devolución introducidos no está en el rango permitido:
    1. El sistema indica el error y finaliza la ejecución.

7a. El sistema comprueba que existe un error en la vida laboral ya que la suma del número de días trabajados en cada trabajo no se corresponde con el total indicado en el propio documento:
    1. El sistema indica el error y finaliza la ejecución.

7b. El sistema comprueba que existe un error en el estado de una propiedad perteneciente al cliente.
    1. El sistema indica el error y finaliza la ejecución.

8a. El sistema comprueba que el cliente es moroso:
    1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8b. El sistema comprueba que el cliente tiene una propiedad embargada:
    1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8c. El sistema comprueba que el cliente está actualmente en paro:
    1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8d. El sistema comprueba que el cliente no supera el Canon francés:
    1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8e. El sistema comprueba que el cliente tiene una propiedad embargada:
    1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8f. El sistema comprueba que el cliente tiene poca experiencia laboral:

1. El sistema indica la denegación por este hecho y finaliza la ejecución.

8g. El sistema comprueba que la relación cantidad solicitada / meses devolución no es correcta:

1. El sistema indica la denegación por este hecho y finaliza la ejecución.

9a. El sistema considera que es necesario un aval:

1. El sistema llama al servicio web que proporciona información sobre el aval.
2. Comprueba que los datos son correctos.
3. Evalúa como válidos los datos.
4. El sistema concede el préstamo al cliente.

9b. El sistema considera que es necesario un aval:

1. El sistema llama al servicio web que proporciona información sobre el aval.
2. Comprueba que los datos son correctos.
3. El sistema se percata de que el aval es moroso.
4. El sistema indica este hecho y deniega el préstamo.

9c. El sistema considera que es necesario un aval:

1. El sistema llama al servicio web que proporciona información sobre el aval.
2. El sistema se percata de que el aval tiene una propiedad embargada.
3. El sistema indica este hecho y deniega el préstamo.

4.2.2. Modelo conceptual de datos

En el modelo conceptual de datos se modela los requisitos de datos de un sistema. El objetivo de esta tarea es identificar y definir las entidades que quedan dentro del ámbito del sistema de información, los atributos de cada entidad, los dominios de los atributos y las relaciones existentes entre las entidades, indicando las cardinalidades mínimas y máximas.

Estas relaciones pueden ser múltiples, recursivas, de explosión e implosión, generalizaciones y agregaciones.

Estas entidades son la representación de los conceptos significativos en el dominio del problema. En la orientación a objetos (entidades que existen en el mundo real y son distinguibles entre ellos), son éstos precisamente los conceptos significativos.
Un modelo conceptual de datos está formado principalmente por:

- **Clases de objetos**: Describe un conjunto de objetos con:
  - Las mismas propiedades
  - Comportamiento común
  - Idéntica relación con otros objetos

    - Semántica común

- **Asociaciones entre clases de objetos**: Es la representación de relaciones entre dos o más objetos
- **Atributos de las clases de objetos**: Propiedad compartida por los objetos de una clase
- **Restricciones de integridad**.

La técnica seguida es la del diagrama de clases, la cual nos permite representar gráficamente todos estos conceptos mencionados anteriormente. En la figura 4.3 podemos observar el diagrama de clases de la composición.

Diagrama de clases - análisis

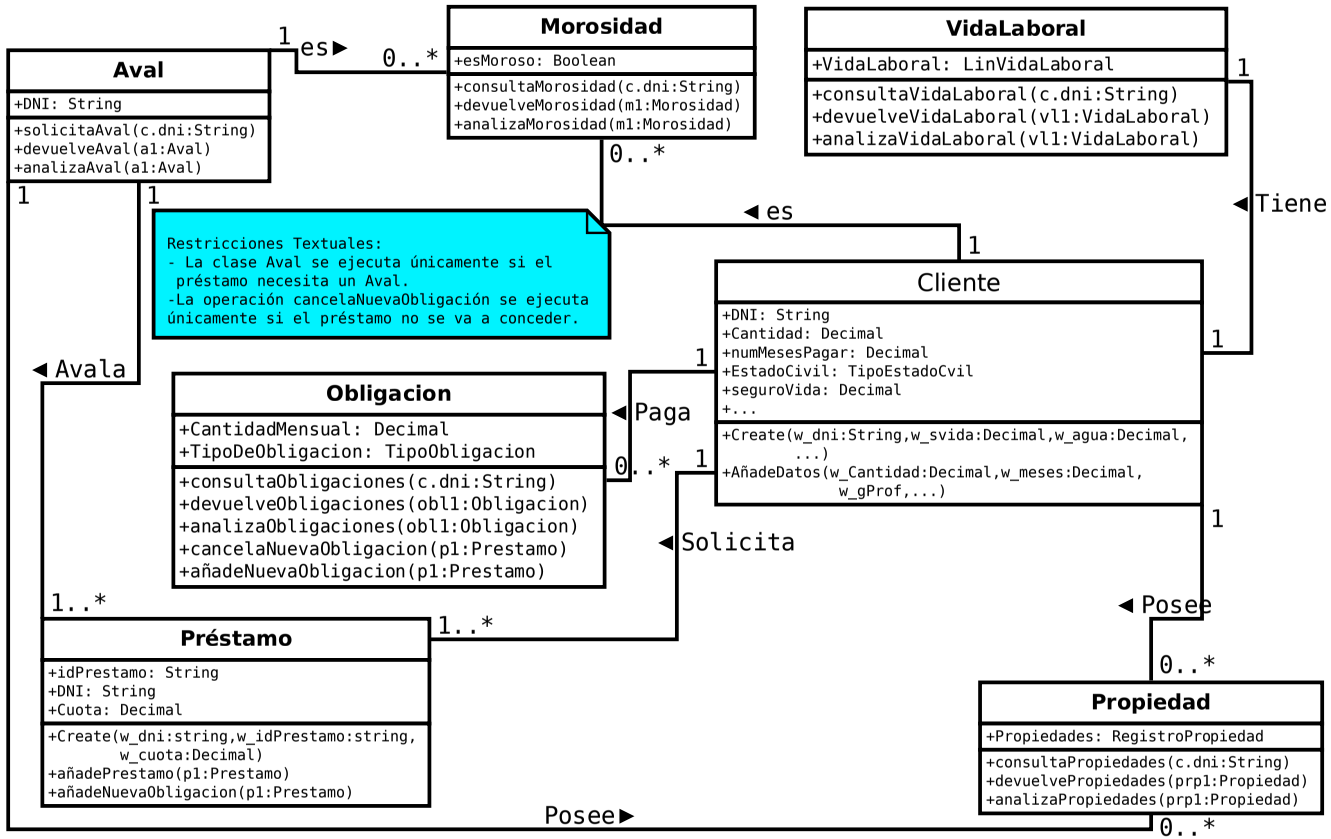

Figura 4.3: Diagrama de Clases - Análisis

4.2.3. Modelo de comportamiento del sistema

El modelado del comportamiento del sistema nos permite obtener la especificación del comportamiento. Las técnicas que emplearemos para el modelado serán los diagramas de secuencia y los contratos de las operaciones. Estas técnicas se deben aplicar a cada uno de los casos de uso que compongan al sistema.

Los diagramas de secuencia:

- Muestran la secuencia de eventos entre los actores y el sistema
- Permiten identificar las operaciones del sistema

Los contratos de las operaciones:

- Describen el efecto de las operaciones del sistema

Caso de uso: Solicitar préstamo personal

Diagrama de secuencia del sistema

En la figura 4.4 podemos observar el diagrama de secuencia del sistema.

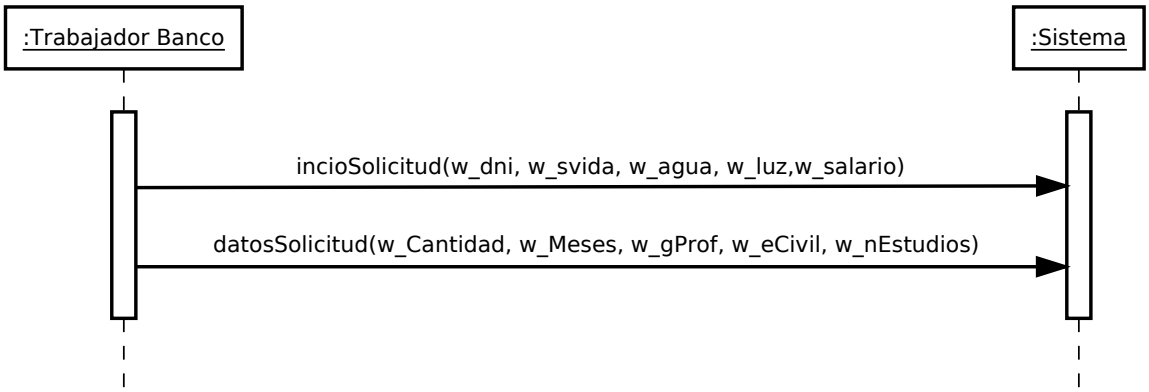

Figura 4.4: Diagrama de Secuencia del Sistema

Contrato de las operaciones

Nombre de la operación: inicioSolicitud(w_dni, w_svida, w_agua, w_luz, w_salario)
Responsabilidades: Se desea iniciar el proceso de solicitud de un préstamo personal.
Precondición:Todos los datos son correctos.
Postcondición:Se crea una instancia C de la clase Cliente con los datos:
C.dni = w_dni, C.segurovida=w_svida, C.luz=w_luz , C.agua=w_agua, C.salario=w_salario.
Se crea un objeto P de la clase Préstamo con valor aleatorio generado automáticamente
al atributo P.idPrestamo y se asocia al objeto C con el valor P.dni = C.dni.

Nombre de la operación: datosSolicitud(w_Cantidad, w_Meses, w_gProf, w_eVivil, w_nEstudios)
Responsabilidades: Se introducen en el sistema el resto de datos que debe proporcionar el cliente.
Precondición:Existe un préstamo en curso.
Postcondición:Se le asigna el resto de atributos a los datos del cliente a la instancia C de la clase Cliente: C.cantidad=w_Cantidad, C.meses=w_Meses, C.grupoProfesion=w_gProf, C.civil=w_eCivil, C.estudios=w_nEstudios.

4.3. Diseño del sistema

El objetivo del proceso de Diseño del Sistema de Información (DSI) es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información.

El diseño del software forma parte del núcleo técnico del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. A partir del análisis y especificación de los requisitos del software, el diseño es la primera de las tres actividades técnicas (diseño, codificación y prueba) que hay que realizar para construir y verificar el software.

4.3.1. Diseño de clases y asociaciones

En la figura 4.5 se muestra el diseño de clases y asociaciones.

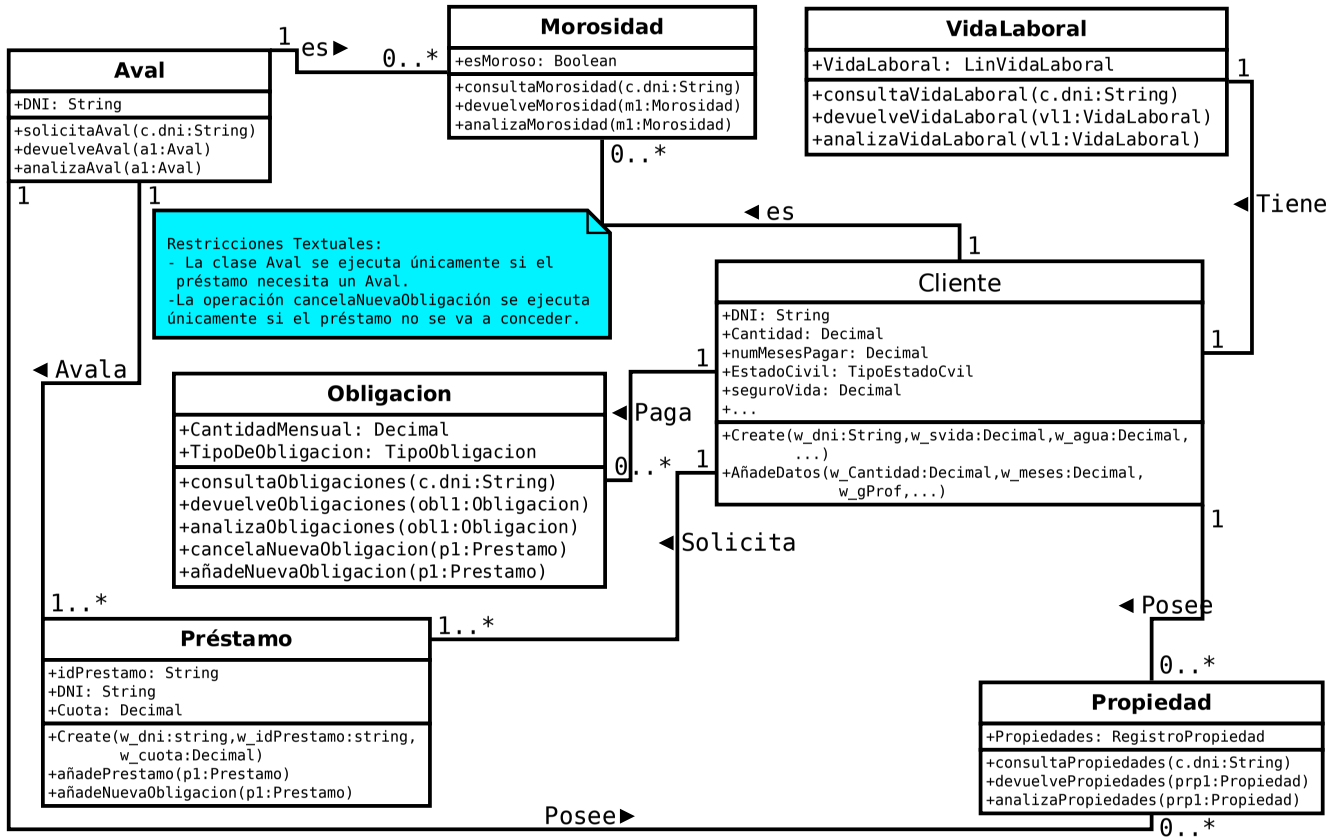

Figura 4.5: Diagrama de clases - diseño

4.3.2. Diagramas de interacción

En ese apartado se describe cómo los objetos colaboran entre sí para realizar cierta actividad es decir, interaccionar. En la interacción un conjunto de mensajes son intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito. Un mensaje es una especificación de una comunicación entre objetos que transmite información, con la expectativa de desencadenar una actividad.

Con los diagramas representamos escenarios y en cada uno de ellos se incluye:

- Objetos y su línea de vida.
- Mensajes.
- Información de control: condiciones y marcas de iteración.
- Indicar el objeto devuelto por el mensaje: return.

En la figura 4.6 observamos el diagrama de interacción de la composición.

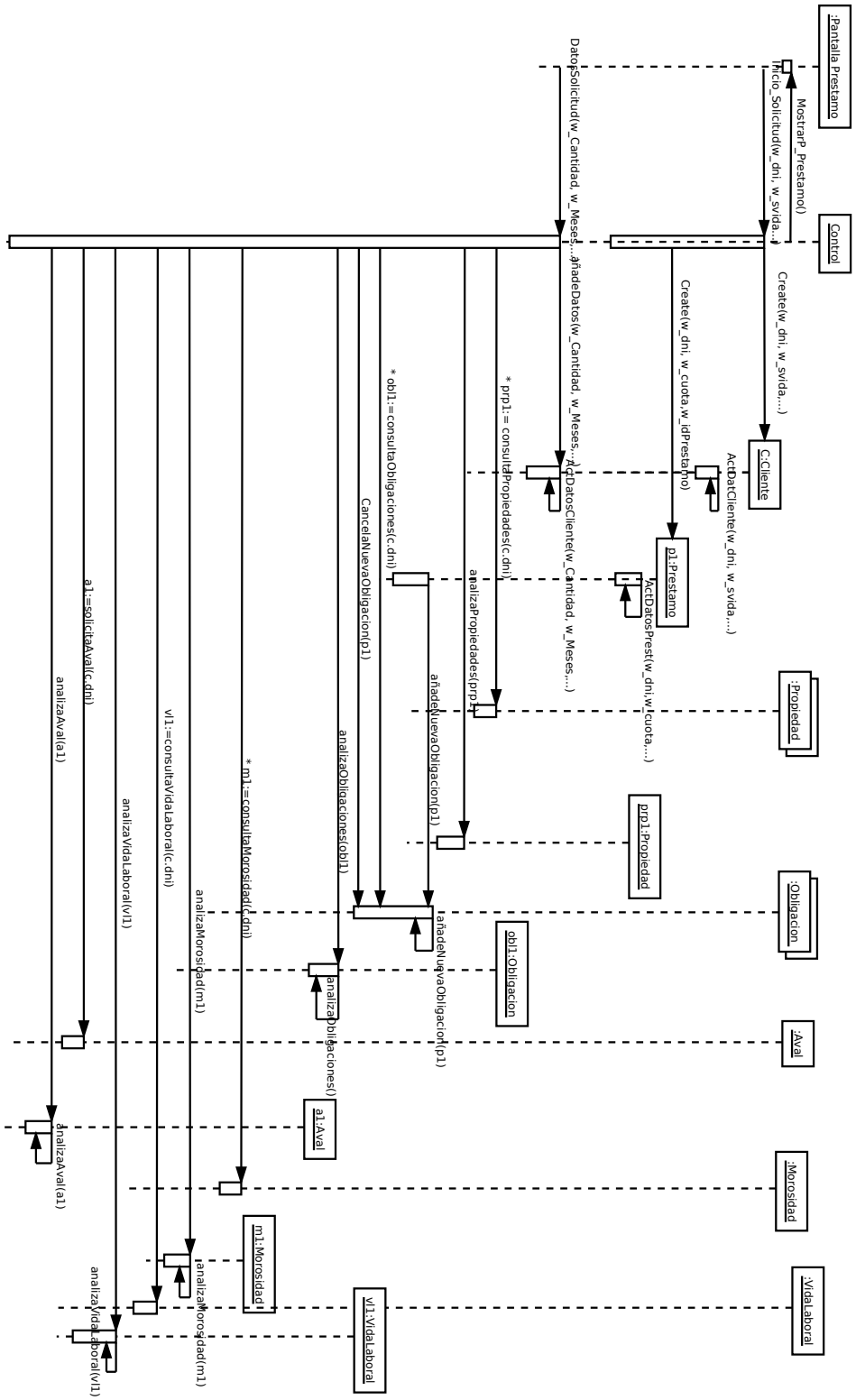

Figura 4.6: Diagrama de interacción

4.3.3. Diagrama de clases de diseño (clases)

Del diagrama de interacción realizado anteriormente, se pueden obtener nuevas clases, asociaciones y operaciones que debemos añadir al diagrama de clases conceptual que teníamos en la etapa de análisis del sistema.

Clases nuevas: clases de control y clases de la interfaz de usuario (simplificadas en clases Pantalla-Prestamo):

- Pantalla Inicial
- SistemaControl
- PrestamoControl
- Pantalla Prestamo

Colaboraciones entre objetos

- Pantalla inicial y Sistema Control
- Sistema Control y Prestamo Control
- Pantalla Prestamo y Prestamo Control
- Prestamo Control y Cliente
- Prestamo Control y Prestamo
- Prestamo Control y Propiedad
- Prestamo Control y Obligaciones
- Prestamo Control y Morosidad
- Prestamo Control y Vida Laboral
- Préstamo y Obligación

4.3.4. Diagrama de clases de diseño (operaciones)

Clase Sistema Control

- Operación que se ejecuta al hacer *new.SistemaControl*:
  - Invoca a la operación *Mostrar Pantalla-inicial* de la clase *PantallaInicial*.
- Operación *Inicio_Prestamo()*:
  - Invoca a la operación *Create()* de la clase *PrestamoControl*.

Clase PantallaInicial

- Operación *Mostrar Pantalla-Inicial*:
  - Muestra la pantalla inicial de la aplicación.

Clase Prestamo Control

- Operación que se ejecuta al hacer *new.PrestamoControl*:
  - Invoca a la operación *Mostrar Pantalla Prestamo* de la clase *Pantalla Prestamo*.
- Operación *InicioSolicitud(w_dni, w_svida, w_agua, w_luz, w_salario)*
  - Invoca a la operación *Create(w_dni, w_svida, w_agua, w_luz, w_salario)* de la clase *Cliente*.
  - Invoca a la operación *Create(w_dni, w_cuota,w_idPrestamo)*. El valor de *w_idPrestamo* es generado automáticamente por el sistema.
  - Invoca a la operación añadeNuevaObligación(p1) de la clase *Prestamo*.
- Operación *datosSolicitud(w_Cantidad, w_meses, w_gProf, w_eCivil, w_nEstudios)*
  - Invoca a la operación *añadeDatos(w_Cantidad, w_meses, w_gProf, w_eCivil, w_nEstudios)* del objeto *C(Cliente)*.
  - Invoca a la operación *consultaPropiedades(c.dni)* de la clase *Propiedades*.
  - Invoca a la operación *analizaPropiedades(prp1)* del objeto prp1 *(Propiedades)*.
  - Invoca a la operación *consultaObligaciones(C.dni)* de la clase *Obligacion*.
  - Invoca a la operación *analizaObligaciones(hip1)* del objeto obl1 *(Obligacion)*.
  - Invoca a la operación *consultaMorosidad(C.dni)* de la clase *Morosidad*.
  - Invoca a la operación *analizaMorosidad(m1)* del objeto m1 *(Morosidad)*.
  - Invoca a la operación *consultaVidaLaboral(C.dni)* de la clase *VidaLaboral*.
  - Invoca a la operación *analizaVidaLaboral(vl1)* del objeto vl1 *(VidaLaboral)*.
  - Invoca a la operación *cancelaNuevaObligacion(p1)* de la clase *Obligación*.

Clase Cliente

- Operación *Create(w_dni, w_svida, w_agua, w_luz, w_salario)*(constructor).
  - Crea un objeto C de la clase *Cliente* con *dni=w_dni, segurovida=w_svida, agua=w_agua, luz=w_luz, salario=w_salario*.
- Operación *añadeDatos(w_Cantidad, w_meses, w_gProf, w_eCivil, w_nEstudios)*.
  - Modifica el valor de los atributos del objeto C con los valores *grupoProfesion=w_gProf, estadoCivil=w_eCivil, meses=w_meses, cantidad=w_Cantidad, nivelEstudios=w_nEstudios*.

Clase Prestamo

- Operación *Create(w_dni, w_cuota,w_idPrestamo)(constructor)*
  - Crea un objeto p1 de la clase *Prestamo* con *dni=w_dni, cuota=w_cuota, idPrestamo=w_idPrestamo*
- Operación *AñadePrestamo(p1)*
  - Añade el objeto p1 del tipo *Préstamo* a la clase *Préstamo*.

- Operación *añadeNuevaObligación(p1)*
  - Invoca a la operación *añadeNuevaObligaciones* de la clase *Obligación*.

Clase Propiedad

- Operación *consultaPropiedades(c.dni)*
  - Consulta en la clase *Propiedad* las propiedades que pueda tener el cliente C.
- Operación *devuelvePropiedades(prp1)*
  - Devuelve un listado prp1 de propiedades que posee C.
- Operación *analizaPropiedades(prp1)*
  - Comprueba el valor del atributo estado de los objetos de la clase Propiedad que componen al listado prp1.

Clase Obligación

- Operación *consultaObligaciones(c.dni)*
  - Consulta en la clase *Obligación* las obligaciones que pueda tener el cliente C.
- Operación *devuelveObligaciones(obl1)*
  - Devuelve un listado obl1 de obligaciones que posee C.
- Operación *analizaObligaciones(obl1)*
  - Acumula el valor del atributo cuota de cada una de las obligaciones que componen al listado obl1.
- *AñadieNuevaObligacion(p1)*
  - Con el inicio de un nuevo préstamo, por defecto se considerará concedido. Esta operación añade el préstamo al conjunto de obligaciones que tiene el cliente.
- *CancelaNuevaOglibacion(p1)*
  - Si ante cualquiera circunstancia se debe denegar el préstamo, esta operación elimina al préstamo p1 del conjunto de obligaciones del cliente.

Clase Morosidad

- Operación *consultaMorosidad(C.dni)*
  - Consulta en la clase *Morosidad* al cliente C.
- Operación *devuelveMorosidad(m1)*
  - Devuelve un objeto m1 con información sobre la morosidad de C.
- Operación *analizaMorosidad(m1)*
  - Comprueba el valor del atributo *esMoroso* del objeto m1.

Clase VidaLaboral

- Operación *consultaVidaLaboral(C.dni)*

- Consulta en la clase *VilaLaboral* la morosidad del cliente C.

- Operación *devuelveVidaLaboral(vl1)*
  - Devuelve la vida laboral vl1 del cliente C.
- Operación *analizaVidaLaboral(hip1)*
  - Analiza cada uno de los diferentes trabajos(duración, puesto desempeñado, tipo contrato...) que ha tenido el cliente a lo largo de su vida laboral vl1.

Clase Aval

- Operación *solicitaAval(C.dni)*
  - Consulta en la clase *Aval* el aval que tiene el cliente C.
- Operación *devuelveAval(a1)*
  - Devuelve el aval del cliente C.
- Operación *analizaAval(a1)*
  - Analiza al aval del cliente C.

OPERADOR	Tiempo en segundos
ERR	14,834
ECC	20,053
ECN	20,026
ASF	19,051
AIE	2,603
ASI	15,751

Cuadro 4.4: Tiempos de creación de mutantes LoanApproval

4.3.5. Pruebas y comparativa

Esta composición es una extensión de una ya existente cuyo nombre es LoanApprovalDoc. Dado que esta composición no refleja el procedimiento real que se debe realizar al conceder un préstamo, se comenzó completamente una nueva desde el principio. No obstante a continuación se muestra una comparación entre los resultados que se obtienen con LoanApprovalDoc y los que se obtienen con LoanApprovalExtended.

Se recomienda leer previamente la sección "Manual de usuario de GAmer" que se encuentra en el capítulo 8 de este documento.

Comparativa de tiempos

Tiempos de LoanApproval

- Mediciones del tiempo de ejecución de *mutationtool run LoanApprovalDoc.bpts LoanApprovalDoc.bpel*: 4,674 segundos.

- Mediciones del tiempo de ejecución de *mutationtool analyze LoanApprovalDoc.bpel*: 4,318 segundos.

- Mediciones del tiempo de ejecución de *mutationtool applyall LoanApprovalDoc.bpel*: El tiempo total para crear cada uno de los mutantes para cada uno de los operadores de mutación se puede observar en la tabla 4.4. La suma de los valores de la tabla 4.4 es en total 92,32.

En la tabla 4.5 se muestran la suma de tiempos que se han empleado para la comparación de cada operador de mutación. La suma de los valores de la tabla 4.5 es en total 630,95 segundos.

Tiempos de LoanApprovalExtended

OPERADOR	Tiempo en segundos
ERR	202,303
ECC	116,672
ECN	57,435
ASF	17,396
AIE	28,580
ASI	208,565

Cuadro 4.5: Tiempos de Comparación LoanApproval

- Mediciones del tiempo de ejecución de *mutationtool run LoanApprovalExtended.bpts LoanApprovalExtended.bpel*: 35,57 segundos.

- Mediciones del tiempo de ejecución de *mutationtool analyze LoanApprovalExtended.bpel*: 4,3 segundos.

- Mediciones del tiempo de ejecución de *mutationtool applyall LoanApprovalExtended.bpel*: El tiempo total para crear cada uno de los mutantes para cada uno de los operadores de mutación se puede observar en la tabla 4.6. La suma de los valores de la tabla 4.6 es en total 5657,79 segundos.

En la tabla 4.7 se muestran las sumas de tiempos que se han empleado para la comparación de cada operador de mutación. La suma de los valores de la tabla 4.7 es en total 88297,63 segundos.

Comparativa de operadores de mutación utilizados

Ahora compararemos el número de operadores de mutación utilizados así como los porcentajes de mutantes vivos, válidos, etc. En primer lugar tenemos los cuadros en los que aparecen el número de veces que se puede aplicar cada operador de mutación a las composiciones.

El cuadro 4.8 muestra los operadores que se pueden aplicar a LoanApprovalDoc.

El número de mutantes generados son 45. De estos 45, tan sólo uno es “no válido” lo que hace un total de 97,78 % de mutantes válidos. De los válidos, 11 están vivos lo que nos da un porcentaje del 75 % de mutantes muertos. El porcentaje de mutantes repetidos es 0 %.

En el cuadro 4.9 observamos los operadores aplicables a LoanApprovalExtended.

OPERADOR	Tiempo en segundos
ISV	3468,91
EAA	287,83
EEU	7,71
ERR	636,95
ELL	25,87
ECC	138,33
ECN	771,01
AFP	1,64
ASF	38,63
AIS	1,66
AIE	49,96
AWR	3,24
AJC	0,87
ASI	165,21
XMF	20,81
XMC	0,94
XTF	29,56
XER	8,48

Cuadro 4.6: Tiempos de creación de mutantes LoanApprovalExtended

OPERADOR	Tiempo en segundos
ISV	40492,87
EAA	5097,24
EEU	241,07
ERR	13791,98
ELL	624,18
ECC	3718,93
ECN	15964,61
AFP	0 73,72
ASF	1432,84
AIS	50,03
AIE	1704,02
AWR	75,23
AJC	33,40
ASI	3673,68
XMF	504,36
XMC	22,31
XTF	690,45
XER	255,80

Cuadro 4.7: Tiempos de Comparación LoanApprovalExtended

OPERADOR	Posibles aplicaciones
ISV	0
EAA	0
EEU	0
ERR	2
ELL	0
ECC	13
ECN	1
EMD	0
EMF	0
ACI	0
AFP	0
ASF	4
AIS	0
AIE	2
AWR	0
AJC	0
ASI	12
APM	0
APA	0
XMF	0
XMC	0
XMT	0
XTF	0
XER	0
XEE	0

Cuadro 4.8: Operadores de mutación aplicables a LA

OPERADOR	Posibles aplicaciones
ISV	1684
EAA	51
EEU	6
ERR	95
ELL	20
ECC	103
ECN	137
EMD	0
EMF	0
ACI	0
AFP	2
ASF	45
AIS	2
AIE	54
AWR	4
AJC	1
ASI	159
APM	0
APA	0
XMF	24
XMC	1
XMT	0
XTF	32
XER	10
XEE	0

Cuadro 4.9: Operadores de mutación aplicables a LAExtended

El número de mutantes generados son 3374. De estos 3374, 3374 son válidos lo que hace un total de 100 % de mutantes válidos. De los válidos, 1082 están vivos lo que nos da un porcentaje del 67,93 % de mutantes muertos. El porcentaje de mutantes repetidos es 0 %.

4.3.6. Validación

Aumentando la calidad del conjunto de casos de prueba

Una vez creada la composición y obtenido un conjunto de casos de pruebas base, el siguiente objetivo es dotar de una mayor calidad a este conjunto. Para ello haremos uso de la herramienta de mutación *mutationtool*. Al leer el título de este apartado una de las preguntas que le viene a la cabeza al lector es que cómo sabemos si un conjunto de casos de prueba es bueno. No existe un punto exacto a partir del cual se pueda decir que un conjunto de casos de prueba es bueno pero sí podemos ir controlando la evolución del mismo para que cada vez sea mejor. Lo que es evidente es, que a mayor número de mutantes muertos del conjunto de mutantes obtenidos mayor será la calidad del conjunto de casos de prueba con el que estamos trabajando.

El proceso está dividido en una serie de pasos que se comentarán a continuación:

- **Análisis de la composición**. En primer lugar debemos analizar la composición para conocer el "alcance" que tendrá la misma. Al analizarla sabremos el número de mutantes que se generarán para cada operador de mutación.
- **Generación de los mutantes**. A continuación generamos el conjunto de mutantes sobre los que vamos a trabajar.
- **Comparación de mutantes**. El resultado de este paso es la generación de un fichero por cada operador de mutación. El contenido de cada uno es una matriz M_{xy}, donde *x* es el número de mutantes diferentes para el operador de mutación en cuestión e *y* el número de casos de prueba que tenemos en *LoanApprovalExtended.bpts*. El valor que podemos encontrar en M_{ij} es 0 ó 1 en función de si el caso de prueba *j* mata al mutante *i*. Tendremos tan sólo un uno como máximo por cada fila. Si una fila está completamente a cero, el mutante está vivo.

  $$M_{ij} = 0 \vee M_{ij} = 1$$

  donde

  $$i \leq x \wedge j \leq y$$

- **Matando mutantes**. Para matar mutantes debemos en primer lugar localizar dónde se encuentra la mutación en el programa original, para ello utilizaremos *TkDiff*. Con *TkDiff* tendremos a media pantalla el documento original y el mutado y podemos observar qué parte del código ha sido modificada y valorar porqué el resultado de la ejecución del conjunto de casos de prueba es el mismo para el original y el mutante.

Esto puede ser debido a dos factores:

- Fallo en la implementación de la composición.
- No hay un caso de prueba que ejecuta las instrucciones que han sido modificadas.

El primer caso es la peor situación ya que habría que modificar la composición y volver a repetir todo el proceso de generación de mutantes y ejecución de matrices de mutantes vivos y muertos. En el segundo caso debemos crear un caso de prueba el cual ejecute esas instrucciones que han sido modificadas para que ahora sí, el resultado final al ejecutar la composición sea diferente en el caso del *.bpel* original y el mutado. Esto que en principio puede resultar sencillo puede convertirse en un proceso bastante tedioso ya que en ocasiones la ejecución de ciertas instrucciones tan sólo ocurre en determinadas condiciones especiales y llegar a dar con esa condición puede ser muy laborioso.

A continuación se muestra un ejemplo. En el código original tenemos el siguiente conjunto de instrucciones:

```
1          <from>-0.4</from>
2          <to variable="ProfesionEval"/>
3          </copy>
```

Tras aplicar un operador de mutación, el mutante generado era el siguiente:

```
1          <from>0.4</from>
2          <to variable="ProfesionEval"/>
3          </copy>
```

El mutante resultante está vivo. Lo cual quiere decir que el resultado de la composición es el mismo asignando a la variable *ProfesionEval*, tanto 0.4 como -0.4 . Nuestra tarea es crear un caso de prueba el cual sea lo suficientemente “sensible” como para que el resultado final sea distinto variando el valor de *ProfesionEval*.

Ahora debemos buscar un punto dentro del código donde repercuta el valor de dicha variable. En este caso son 2:
El primero:

```
1 <copy>
2          <from>($LaboralEvaluacion + (-1 * $ProfesionEval) +
3     $CivilEval + $EstudiosEval) * $propiedadesEvaluacion</from>
4          <to variable="GlobalEvaluacion"/>
5 </copy>}
```

El segundo:

```
<if>
<condition>$GlobalEvaluacion &lt; 5.01</condition>
        <throw faultName="tns:DenegacionEvalPersonal"/>
</if>
```

Como se puede observar el valor de *ProfesionEval* influye en la variable *GlobalEvaluacion*. Si *GlobalEvaluacion* es menor a 5.01 lanzará una falta que posteriormente finalizará la ejecución de la composición. Así que es aquí donde debemos buscar la diferencia en la ejecución de la composición.

Ahora debemos buscar una combinación de valores para las variables *LaboralEvaluación, CivilEval, EstudioEval* y *propiedadesEvaluacion* para los cuales variando de -0.4 a 0.4 el valor de *ProfesionEval*, el valor de *GlobalEvaluacion* pase de ser mayor a 5.01 (ejecución normal y satisfactoria de la composición) a menor que éste. De esa manera el resultado final será diferente y por tanto el mutante generado estará muerto.

Con estas premisas vamos dando valores a las variables hasta conseguir que el resultado de la composición sea diferente cambiando el valor de *ProfesionEval* de -0,4 a 0,4. Este puede ser un proceso bastante largo ya que por cada ejecución y comprobación de mutante muerto se puede tardar en torno a los 30-35 segundos.

Además los valores de las variables en ocasiones no se asignan directamente, sino que son valores obtenidos de trabajar con otras. Por ejemplo, *LaboralEvaluacion* toma su valor tras operar con las variables *num_dias, Puesto, Contrato* que recibe del *mockup* de la *seguridad social*:

```
<VidaLaboral>
        <ns0:FechaEmision>2010-01-01</ns0:FechaEmision>
        <ns0:TotalDias>425</ns0:TotalDias>
        <ns0:Trabajo>
                <ns0:FechaAlta>2008-09-09</ns0:FechaAlta>
                <ns0:FechaBaja>null</ns0:FechaBaja>
                <ns0:num_dias>425</ns0:num_dias>
                <ns0:Puesto>09</ns0:Puesto>
                 <ns0:Contrato>501</ns0:Contrato>
        </ns0:Trabajo>
</VidaLaboral>}
```

Descripción de la estructura de los casos de prueba

En primer lugar tenemos los datos que proporcionaría el cliente a la entidad. Éstos se pueden dividir en dos grupos:

Un primero en el que podemos encontrar información de su situación personal:

- D.N.I.
- Cantidad solicitada.
- Salario.
- Número de meses en los que devolverla.
- Grupo de profesión al que pertenece.
- Estado civil.
- Nivel de estudios que posee.

Y un segundo grupo referente a gastos mensuales:

- Seguro de vida.
- Seguro del hogar.
- Consumo de agua.
- Consumo de luz.

A partir de aquí tenemos 4 bloques que corresponden con los datos que obtendríamos de los diferentes servicios web contemplados. Hay que recordar que estos servicios web son sustituidos por *mockups*.

Seguridad Social

De ella obtendríamos el informe de vida laboral del cliente. Esta vida laboral tendría la siguiente información (la cual no es inventada, sino que viene reflejada en cualquier informe de Vida laboral real):

- Fecha de emisión del documento.
- Número total de días trabajados por el solicitante durante toda su vida.
- Listado con información de cada uno de los trabajos.
    - Fecha de alta.
    - Fecha de baja.
    - Número de días trabajados.
    - Puesto ocupado.
    - Tipo de contrato.

Se han distinguido 11 puestos y de 6 tipos de contratos, los cuales se pueden ver a continuación. Todos ellos existen realmente como se mencionó anteriormente.
Puestos:

- Ingeniero/Licenciado
- Ingeniero técnico/Perito
- Jefe administrativo de taller
- Ayudante no titulado
- Oficial administrativo
- Subalterno
- Auxiliar administrativo
- Oficial de primera y segunda
- Oficial de tercera especialista
- Peón
- Menor de 18 años

Contratos:

- Indefinido-Tiempo Completo
- Indefinido-Tiempo Parcial
- Duración Determinada-Tiempo Completo
- Duración Determinada-Tiempo Parcial
- Temporal-Tiempo Completo
- Temporal-Tiempo Parcial

A.S.N.E.F.

Con este servicio web sabremos si el cliente es moroso. Así que únicamente nos proporcionaría un sí (*true*) o no (*false*) en función de si el cliente es o no moroso.

Banco de España

La información proporcionada por la CIRBE (Central de Información de Riesgo del Banco de España) consiste en un listado de cada uno de las obligaciones que posee un cliente. Se han distinguido dos tipos de obligación:

- Préstamo
- Hipoteca

Por cada obligación tendremos una dupla con la cantidad mensual que debe abonar mensualmente para reembolsar la obligación y el tipo de la misma.

Ministerio de Justicia

A través del Registro de la Propiedad conoceremos las propiedades que posee el cliente así como el estado de las mismas. Los estados distinguidos son los siguientes:

- Libre de carga. En el caso de que la propiedad esté ya pagada en su totalidad.
- Hipotecada con nosotros. Es decir que la propiedad tiene una hipoteca y ésta la tiene en nuestra entidad. Esto será sumamente importante ya que en caso de ser necesario, se podrá proceder al embargo de la misma.
- Hipotecada con otra entidad. Lo cual es negativo ya que no podremos proceder al embargo.
- Embargada. Si el cliente tiene una propiedad embargada, se le denegará el préstamo.

Ocasionalmente en función del caso de prueba, tendremos un quinto bloque que representa los datos referentes al aval. En él tendremos el D.N.I. del mismo, que nos servirá para llamar a los distintos servicios web para obtener sus datos. Del aval evaluaremos únicamente sus propiedades y su morosidad.

Descripción textual de los casos de prueba

El número de casos de pruebas que se han realizado para la composición es de 40. Con ellos, se ha tratado de abarcar un gran número de situaciones que se pueden dar cuando se concede un préstamo personal. Además hay que mencionar que muchos casos de pruebas son similares entre sí pero con pequeñas variaciones con el fin de matar a algunos mutantes vivos en particular.

Estos casos de prueba suelen tener en su título un contenido similar a "m02-44-01" por ejemplo, esto quiere decir que el caso de prueba en cuestión se ha realizado para matar al mutante generado al aplicar el operador de mutación EAA (número 2 de la lista de operadores de mutación) a la instancia número 44 donde se podía aplicar el mismo.

A continuación se realiza una breve descripción textual de los mismos.

1. **testCase11- Fault Grupo Prof. Erroneo**.
   Situación tratada: En los datos proporcionados por el cliente, indica un grupo de profesión que no existe. En tal caso, la composición detecta el error y lanza una falta que trata al mismo. La composición cancela la ejecución y devuelve un mensaje indicado el error cometido. Errores similares son los cometidos en los siguientes casos de prueba.
2. **testCase11- Fault Nivel EStudios. Erroneo**.
   El nivel de estudios no es válido.
3. **testCase11- Fault Estado civil. Erroneo**.
   El estado civil no es válido.
4. **testCase11- Fault Estado Propiedad. Erroneo**.
   En este caso de prueba, en el mensaje recibido desde el servicio web del Registro de la Propiedad, el estado de una de las propiedades no es válido. Al igual que en los casos anteriores, se indica el error.
5. **testCase8-b- Denegado. Evaluacion Global-m02-44-01**.
   En este caso no hay ningún dato erróneo. La composición se ejecuta con normalidad casi en su totalidad hasta que realiza la evaluación global de todos los datos ofrecidos por el cliente y los diferentes servicios web y detecta que los valores obtenidos no superan los mínimos para conceder con garantías el préstamo. Se deniega la conceción y se le comunica la razón al cliente.
6. **testCase1-a-Concedido-Avalista-m14-10-01-ContratoTemporal**.
   El préstamo es concedido con la ayuda de un avalista. Éste se hace necesario porque el cliente tiene un contrato de trabajo de tipo temporal.
7. **testCase1-a-Concedido-Avalista-m14-08-01-ContratoTemporal**.
   Situación similar a la anterior pero matando a un mutante diferente.
8. **testCase0-a-m14-23-25-etc-01-Concedido**.
   En este caso de prueba (y los 10 siguientes), el préstamo es concedido al cliente

por lo que se debe indicar las condiciones de reembolso del mismo. Las diferencias entre el actual caso y los siguientes son mínimos.

9. **testCase0-a-m14-35-01-Concedido**.
   La diferencia con el caso de prueba anterior radica en el nivel de estudios. En este caso el nivel es de "primarios no finalizados".
10. **testCase0-a-m14-39-01-Concedido**.
    Aquí el nivel de estudios es "ingeniería técnica".
11. **testCase0-a-m14-49-01-Concedido**.
    El cliente tiene un estado civil diferente. Está "separado".
12. **testCase0-a-m14-47-01-Concedido**.
    El estado civil es "viudo".
13. **testCase0-a-m14-07-01-Concedido**.
14. **testCase0-m14-06-Concedido**.
    El grupo de profesión al que pertenece el cliente es el tercero.
15. **testCase0-a-m03-02-01-Concedido**.
    El grupo de profesión al que pertenece el cliente es el segundo.
16. **testCase0-a-m03-03-01-Concedido**.
    El cliente tiene trabajo en la actualidad. El tipo de contrato que posee es "Duración Determinada Tiempo Parcial".
17. **testCase0-a-m03-04-01-Concedido**.
    El grupo de profesión es el 4.
18. **testCase0-a-m03-05-01-Concedido**.
    El grupo de profesión es el 5.
19. **testCase-Denegado-El Avalista tiene Embargos**.
    El cliente no cumple los requisitos para concederle el préstamo. La razón es que no llega al mínimo de días que debe haber trabajado a lo largo de su vida laboral (180). Por ello es necesaria la figura del aval. Una vez con los datos del aval, se evalúa al mismo y se detecta que éste tiene embargos a su nombre. Ya que no podemos aceptar un avalista con embargos, se deniega el préstamo y se le indica al cliente.
20. **testCase-Denegado-El Avalista es Moroso**.
    Nos encontramos con una situación similar a la anterior, pero con la salvedad de que el avalista no tiene embargos pero sí es moroso. Al igual que antes, se deniega el préstamo y se comunica la razón de denegación.
21. **testCase0-a-Concedido**.
    Tanto en el actual como en los siguientes 4 casos de prueba, tenemos diferentes valores tanto para los datos del cliente como para los datos obtenidos de los diferentes servicios web con el fin de abarcar diferentes situaciones posibles.
22. **testCase0-b-Concedido**.
23. **testCase0-c-Concedido**.

24. **testCase0-d-Concedido**.
25. **testCase0-e-Concedido**.
26. **testCase1-a-Concedido-Avalista-ContratoTemporal**.
    El cliente tiene un contrato temporal por lo que se hace necesario un avalista. Se evalúan los datos del aval y todos son correctos por lo que se le concede el préstamo al cliente.
27. **testCase1-b-Concedido-Avalista-PocaExperiencia**.
    El cliente ha trabajado poco a lo largo de su vida laboral y es necesario el avalista. Se evalúan los datos del aval y todos son correctos por lo que se le concede el préstamo al cliente.
28. **testCase2 -a- Denegado - Propiedad Embargada**.
    Se observa a través de los datos del Registro de la Propiedad que el cliente tiene una propiedad embargada. Se deniega inmediatamente el préstamo y se comunica la razón.
29. **testCase2 -b- Denegado - Moroso y Propiedad Embargada**.
    En este caso de prueba, el cliente además de tener una propiedad embargada, es moroso. Son 2 condiciones suficientes como para denegarle el préstamo.
30. **testCase3 - Denegado - Morosidad**. A través de la ASNEF observamos que el cliente es moroso. Se deniega por tanto el préstamo al cliente.
31. **testCase4 - Denegado - NumMeses/Cantidad**.
    El cliente al solicitar su préstamo debe indicar la cantidad solicitada y el número de meses en los que desea reembolsar el mismo. Estas cantidades deben guardar una relación razonable entre sí ya que no se pueden dar situaciones como la siguiente: "Solicitar un préstamo de 3.000 Euros y desear reembolsarlo en 10 años". Para ello se establece un coeficiente entre ambas que se debe cumplir. En este caso de prueba este coeficiente no pasa el umbral establecido como mínimo establecido y se deniega el préstamo.
32. **testCase5 - Denegado En paro**.
    El cliente actualmente está en paro. Se deniega el préstamo.
33. **testCase6 -a- Denegado Canon Frances**.
    El concepto de “Canon Francés” ya se explicó con anterioridad. En este caso de prueba el cliente no supera dicho Canon a sobrepasar los máximos establecidos como tope de seguridad.
34. **testCase6 -b - Denegado Canon Frances**.
35. **testCase7- Denegado. Trabajado Poco**.
    En este caso el cliente sí tiene suficiente experiencia laboral pero la relación de días trabajados a lo largo de su vida laboral respecto a los días que podría haber trabajado no es buena así que considera al cliente como inestable desde el punto de vista laboral. Se deniega el préstamo y se comunica la razón.
36. **testCase8- Denegado . Evaluacion Global**.

37. **testCase9- Fault Numero Meses Reembolso**.
    Situación de error. El número de meses en los que se debe reembolsar el préstamo no puede exceder un tope. En este caso el excede este tope y se le comunica.
38. **testCase10-a- Fault Cantidad No válida**.
    La cantidad solicitada por el cliente debe estar entre 3.000 Euros y 60.000 Euros. Si sobrepasa estos límites no será un préstamo válido y se comunicará el error. En este caso la cantidad es inferior a 3.000.
39. **testCase10-b- Fault Cantidad No válida**.
40. **testCase-Error Vida laboral**.
    En este caso de prueba la cantidad es mayor a 60.000 Euros. La suma del número de días trabajados en cada trabajo de manera individual debe coincidir con el valor de días totales que aparece en el informe de vida laboral. En caso contrario habría algún error y se cancelaría el proceso. En este caso nos encontramos con esta situación por lo que se detiene la ejecución de la composición y se informa del error.

Algunos ejemplos concretos de casos de prueba

A continuación se validará que la ejecución por parte de LoanApprovalExtended es correcta para algunos casos de pruebas genéricos.

Caso de prueba #1. Resultado esperado: Concedido

Datos obtenidos del cliente como entrada de la composición.

```
        <ns0:dni>'31709567L'</ns0:dni>
        <ns0:cantidad>3000</ns0:cantidad>
        <ns0:numMesesPagar>12</ns0:numMesesPagar>
        <ns0:grupoProfesion>Grupo3</ns0:grupoProfesion>
        <ns0:EstadoCivil>Soltero</ns0:EstadoCivil>
        <ns0:Estudios>PrimariosNoFinalizados</ns0:Estudios>
        <ns0:seguroVida>0</ns0:seguroVida>
        <ns0:seguroCasa>0</ns0:seguroCasa>
        <ns0:gastoAgua>35</ns0:gastoAgua>
        <ns0:gastoLuz>25</ns0:gastoLuz>
        <ns0:salario>955</ns0:salario>
```

Y para los siguientes conjuntos de datos obtenidos de los diferentes Servicios Web:

```
<!--SW_SSOCIAL-->
        <VidaLaboral>
                <ns0:FechaEmision>2010-01-01</ns0:FechaEmision>
                <ns0:TotalDias>425</ns0:TotalDias>
                <ns0:Trabajo>
                        <ns0:FechaAlta>2008-09-09</ns0:FechaAlta>
                        <ns0:FechaBaja>null</ns0:FechaBaja>
                        <ns0:num_dias>425</ns0:num_dias>
                        <ns0:Puesto>09</ns0:Puesto>
                        <ns0:Contrato>401</ns0:Contrato>
                </ns0:Trabajo>
        </VidaLaboral>

<!--SW_ASNEF-->
        <esMoroso>false</esMoroso>


<!--SW_CIRBE-->
        <obligaciones>
        <ns0:obl>
                <ns0:CantidadMensual>150</ns0:CantidadMensual>
                <ns0:TipoDeObligacion>Prestamo</ns0:TipoDeObligacion
                   >
```

```
        </ns0:obl>
        <ns0:obl>
                <ns0:CantidadMensual>75</ns0:CantidadMensual>
                <ns0:TipoDeObligacion>Prestamo</ns0:TipoDeObligacion
                   >
        </ns0:obl>
        </obligaciones>

<!--SW_REGPROPIEDAD-->
        <propiedades>
                <ns0:NotaSimple>
                        <ns0:Estado>Ninguna</ns0:Estado>
                </ns0:NotaSimple>
        </propiedades>
```

El conjunto de datos de salida esperado es el siguiente:

```
        <tes:condition>
                <tes:expression>dictamen</tes:expression>
                <tes:value>'true'</tes:value>
        </tes:condition>
        <tes:condition>
                <tes:expression>razon</tes:expression>
        <tes:value>'Cumple todos los requisitos'</tes:value>
        </tes:condition>
```

Y observando la salida en el fichero de salida.xml efectivamente podemos ver que el resultado obtenido es el esperado:

```
<tes:receiveCondition name="Receive Condition" result="PASSED"
message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
        <tes:state name="Expression">dictamen</tes:state>
        <tes:state name="Value">'true'</tes:state>
        <tes:condition>
                <tes:expression>dictamen</tes:expression>
                <tes:expectedValue>'true'</tes:expectedValue>
                <tes:actualValue xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        </tes:condition>
</tes:receiveCondition>
<tes:receiveCondition name="Receive Condition" result="PASSED"
message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
```

```
18          <tes:state name="Expression">razon</tes:state>
19          <tes:state name="Value">'Cumple todos los requisitos'</tes:
               state>
20          <tes:condition>
21                  <tes:expression>razon</tes:expression>
22                  <tes:expectedValue>'Cumple todos los requisitos'</
                       tes:expectedValue>
23                  <tes:actualValue xsi:nil="true"
24  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
25          </tes:condition>
26  </tes:receiveCondition>
```

Caso de prueba #2. Resultado esperado: Concedido con Aval
Para el siguiente conjunto de datos de entrada:

```
        <ns0:dni>'31709567L'</ns0:dni>
        <ns0:cantidad>3000</ns0:cantidad>
        <ns0:numMesesPagar>12</ns0:numMesesPagar>
        <ns0:grupoProfesion>Grupo1</ns0:grupoProfesion>
        <ns0:EstadoCivil>Soltero</ns0:EstadoCivil>
        <ns0:Estudios>SinEstudios</ns0:Estudios>
        <ns0:seguroVida>10</ns0:seguroVida>
        <ns0:seguroCasa>10</ns0:seguroCasa>
        <ns0:gastoAgua>10</ns0:gastoAgua>
        <ns0:gastoLuz>10</ns0:gastoLuz>
        <ns0:salario>1000</ns0:salario>
```

Y para los siguientes conjuntos de datos obtenidos de los diferentes Servicios Web:

```
<!--SW_SSOCIAL-->
        <VidaLaboral>
                <ns0:FechaEmision>2010-04-04</ns0:FechaEmision>
                <ns0:TotalDias>500</ns0:TotalDias>
                <ns0:Trabajo>
                        <ns0:FechaAlta>2008-09-08</ns0:FechaAlta>
                        <ns0:FechaBaja>1975-09-09</ns0:FechaBaja>
                        <ns0:num_dias>300</ns0:num_dias>
                        <ns0:Puesto>11</ns0:Puesto>
                        <ns0:Contrato>501</ns0:Contrato>
                </ns0:Trabajo>
                <ns0:Trabajo>
                        <ns0:FechaAlta>1975-09-09</ns0:FechaAlta>
                        <ns0:FechaBaja>null</ns0:FechaBaja>
                        <ns0:num_dias>200</ns0:num_dias>
                        <ns0:Puesto>05</ns0:Puesto>
                        <ns0:Contrato>508</ns0:Contrato>
                </ns0:Trabajo>
        </VidaLaboral>

<!--SW_ASNEF-->
<!--Para el cliente:-->
        <esMoroso>false</esMoroso>

<!--Para el aval:-->
\begin{lstlisting}
        <esMoroso>false</esMoroso>

<!--SW_CIRBE-->
```

```
30          <obligaciones>
31          <ns0:obl>
32                  <ns0:CantidadMensual>150</ns0:CantidadMensual>
33                  <ns0:TipoDeObligacion>Prestamo</ns0:TipoDeObligacion
                       >
34          </ns0:obl>
35          </obligaciones>
36
37
38  <!--SW_REGPROPIEDAD-->
39  <!--Para el cliente:-->
40  \begin{lstlisting}
41          <propiedades>
42                  <ns0:NotaSimple>
43                          <ns0:Estado>HipotecadaOtraEntidad</ns0:
                               Estado>
44                  </ns0:NotaSimple>
45          </propiedades>
46
47  <!--Para el aval:-->
48          <propiedades>
49                  <ns0:NotaSimple>
50                          <ns0:Estado>LibreDeCarga</ns0:Estado>
51          </ns0:NotaSimple>
52
53
54  <!--SW_AVAL-->
55          <ns0:dni>'33333333K'</ns0:dni>
```

El conjunto de datos de salida esperado es el siguiente:

```
1           <tes:condition>
2           <tes:expression>dictamen</tes:expression>
3           <tes:value>'true'</tes:value>
4           </tes:condition>
5           <tes:condition>
6           <tes:expression>razon</tes:expression>
7           <tes:value>'Cumple todos los requisitos'</tes:value>
8           </tes:condition>
```

Y observando la salida en el fichero de salida.xml efectivamente podemos ver que el resultado obtenido es el esperado:

```
1   <tes:receiveCondition name="Receive Condition" result="PASSED"
2   message="Passed">
3           <tes:state name="Status Code">PASSED</tes:state>
```

```
        <tes:state name="Status Message">Passed</tes:state>
        <tes:state name="Expression">dictamen</tes:state>
        <tes:state name="Value">'true'</tes:state>
        <tes:condition>
                <tes:expression>dictamen</tes:expression>
                <tes:expectedValue>'true'</tes:expectedValue>
                <tes:actualValue xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        </tes:condition>
</tes:receiveCondition>
<tes:receiveCondition name="Receive Condition" result="PASSED"
message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
        <tes:state name="Expression">razon</tes:state>
        <tes:state name="Value">'Cumple todos los requisitos'</tes:
           state>
        <tes:condition>
                <tes:expression>razon</tes:expression>
                <tes:expectedValue>'Cumple todos los requisitos'</
                   tes:expectedValue>
                <tes:actualValue xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        </tes:condition>
</tes:receiveCondition>
```

Caso de prueba #3. Resultado esperado: Denegado por propiedad embargada
Para el siguiente conjunto de datos de entrada:

```
<ns1:datos>
<ns0:dni>'31709567L'</ns0:dni>
<ns0:cantidad>3000</ns0:cantidad>
<ns0:numMesesPagar>12</ns0:numMesesPagar>
<ns0:grupoProfesion>Grupo1</ns0:grupoProfesion>
<ns0:EstadoCivil>Soltero</ns0:EstadoCivil>
<ns0:Estudios>SinEstudios</ns0:Estudios>
<ns0:seguroVida>10</ns0:seguroVida>
<ns0:seguroCasa>10</ns0:seguroCasa>
<ns0:gastoAgua>10</ns0:gastoAgua>
<ns0:gastoLuz>10</ns0:gastoLuz>
<ns0:salario>1000</ns0:salario>
</ns1:datos>
```

Y para los siguientes conjuntos de datos obtenidos de los diferentes Servicios Web:

```
<!--SW_ASNEF-->
        <esMoroso>false</esMoroso>

<!--SW_CIRBE-->
<obligaciones>
        <ns0:obl>
                <ns0:CantidadMensual>0</ns0:CantidadMensual>
                <ns0:TipoDeObligacion>Ninguno</ns0:TipoDeObligacion>
        </ns0:obl>
</obligaciones>


<!--SW_REGPROPIEDAD-->
<propiedades>
        <ns0:NotaSimple>
<ns0:Estado>Embargada</ns0:Estado>
        </ns0:NotaSimple>
</propiedades>
```

El conjunto de datos de salida esperado es el siguiente:

```
<tes:condition>
        <tes:expression>dictamen</tes:expression>
        <tes:value>'false'</tes:value>
</tes:condition>
<tes:condition>
```

```
        <tes:expression>razon</tes:expression>
        <tes:value>'Tiene una propiedad embargada'</tes:value>
</tes:condition>
```

Y observando la salida en el fichero de salida.xml efectivamente podemos ver que el resultado obtenido es el esperado:

```
<tes:receiveCondition name="Receive Condition" result="PASSED"
 message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
    <tes:state name="Status Message">Passed</tes:state>
    <tes:state name="Expression">dictamen</tes:state>
    <tes:state name="Value">'false'</tes:state>
    <tes:condition>
        <tes:expression>dictamen</tes:expression>
        <tes:expectedValue>'false'</tes:expectedValue>
        <tes:actualValue xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        </tes:condition>
</tes:receiveCondition>
<tes:receiveCondition name="Receive Condition" result="PASSED"
 message="Passed">
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
        <tes:state name="Expression">razon</tes:state>
        <tes:state name="Value">'Tiene una propiedad embargada'</tes
           :state>
        <tes:condition>
        <tes:expression>razon</tes:expression>
        <tes:expectedValue>'Tiene una propiedad embargada'</tes:
           expectedValue>
        <tes:actualValue xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
        </tes:condition>
</tes:receiveCondition>
```

Caso de prueba #4. Resultado esperado: Denegado por aval moroso
Para el siguiente conjunto de datos de entrada:

```
<ns0:dni>'31709567L'</ns0:dni>
<ns0:cantidad>3000</ns0:cantidad>
<ns0:numMesesPagar>12</ns0:numMesesPagar>
<ns0:grupoProfesion>Grupo1</ns0:grupoProfesion>
<ns0:EstadoCivil>Soltero</ns0:EstadoCivil>
<ns0:Estudios>SinEstudios</ns0:Estudios>
<ns0:seguroVida>10</ns0:seguroVida>
<ns0:seguroCasa>10</ns0:seguroCasa>
<ns0:gastoAgua>10</ns0:gastoAgua>
<ns0:gastoLuz>10</ns0:gastoLuz>
<ns0:salario>1000</ns0:salario>
```

Y para los siguientes conjuntos de datos obtenidos de los diferentes Servicios Web:

```

<!--SW_SSOCIAL-->
<VidaLaboral>
<ns0:FechaEmision>2010-12-12</ns0:FechaEmision>
<ns0:TotalDias>179</ns0:TotalDias>
<ns0:Trabajo>
<ns0:FechaAlta>2010-02-02</ns0:FechaAlta>
<ns0:FechaBaja>null</ns0:FechaBaja>
<ns0:num_dias>179</ns0:num_dias>
<ns0:Puesto>05</ns0:Puesto>
<ns0:Contrato>100</ns0:Contrato>
</ns0:Trabajo>
</VidaLaboral>


<!--SW_ASNEF-->
<!--Para el cliente:-->
        <esMoroso>false</esMoroso>

<!--Para el aval:-->
        <esMoroso>true</esMoroso>


<!--SW_CIRBE-->
<obligaciones>
        <ns0:obl>
        <ns0:CantidadMensual>150</ns0:CantidadMensual>
        <ns0:TipoDeObligacion>Prestamo</ns0:TipoDeObligacion>
        </ns0:obl>
</obligaciones>
```

```
31
32
33  <!--SW_REGPROPIEDAD-->
34  <!--Para el cliente:-->
35          <propiedades>
36          <ns0:NotaSimple>
37          <ns0:Estado>LibreDeCarga</ns0:Estado>
38          </ns0:NotaSimple>
39          </propiedades>
40
41  <!--Para el aval:-->
42          <propiedades>
43          <ns0:NotaSimple>
44          <ns0:Estado>LibreDeCarga</ns0:Estado>
45          </ns0:NotaSimple>
46          </propiedades>
47
48
49  <!--SW_AVAL:-->
50  <ns0:dni>'33333333K'</ns0:dni>
```

El conjunto de datos de salida esperado es el siguiente:

```
1  <tes:condition>
2  <tes:expression>dictamen</tes:expression>
3  <tes:value>'false'</tes:value>
4  </tes:condition>
5  <tes:condition>
6  <tes:expression>razon</tes:expression>
7  <tes:value>'Su aval es moroso'</tes:value>
8  </tes:condition>
```

Y observando la salida en el fichero de salida.xml efectivamente podemos ver que el resultado obtenido es el esperado:

```
1  <tes:receiveCondition name="Receive Condition" result="PASSED"
2  message="Passed">
3          <tes:state name="Status Code">PASSED</tes:state>
4          <tes:state name="Status Message">Passed</tes:state>
5          <tes:state name="Expression">dictamen</tes:state>
6          <tes:state name="Value">'false'</tes:state>
7                  <tes:condition>
8                  <tes:expression>dictamen</tes:expression>
9                  <tes:expectedValue>'false'</tes:expectedValue>
10                 <tes:actualValue xsi:nil="true"
11 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
```

```
12                  </tes:condition>
13  </tes:receiveCondition>
14  <tes:receiveCondition name="Receive Condition" result="PASSED"
15  message="Passed">
16          <tes:state name="Status Code">PASSED</tes:state>
17          <tes:state name="Status Message">Passed</tes:state>
18          <tes:state name="Expression">razon</tes:state>
19          <tes:state name="Value">'Su aval es moroso'</tes:state>
20          <tes:condition>
21                  <tes:expression>razon</tes:expression>
22                  <tes:expectedValue>'Su aval es moroso'</tes:
                       expectedValue>
23                              <tes:actualValue xsi:nil="true"
24  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
25          </tes:condition>
26  </tes:receiveCondition>
```

5 Resumen

Éste es un Proyecto Fin de Carrera propuesto por el grupo de investigación SPI&FM del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz. Está orientado al área de la investigación ya que tiene como finalidad ser ejecutado en dos herramientas que están siendo desarrolladas por los miembros del grupo en su actual línea de trabajo. Estas herramientas se denominan GAmera y Takuan.

GAmera es un sistema generador automático de mutantes que permite medir la calidad de un conjunto de casos de pruebas para una composición realizada en dicho lenguaje basándose en el análisis de mutaciones. Esta técnica se basa en obtener *mutantes* a partir de un programa original. Un *mutante* no es más que el programa original con alguna pequeña modificación con la finalidad de modelar los fallos más comunes que poseen los programadores al desarrollar alguna aplicación. Cada pequeña modificación está realizada por lo que se conoce como operador de mutación. En SPI&FM se han definido 25 operadores de mutación para WS-BPEL 2.0.

A estos *mutantes* se les aplica el conjunto de casos de prueba de la composición original, si la salida es diferente a la original se dice que el mutante está *muerto*, en caso contrario estará *vivo*. El objetivo es que todos los *mutantes mueran*, si todos están *muertos* significa que tenemos un conjunto de casos de prueba excelente. Si alguno está *vivo* puede significar dos cosas:

- Error en la composición.
  En este caso se debe modificar la composición original para corregir el fallo.
- Conjunto de casos de prueba pobre.
  Se debe aumentar el conjunto de casos de prueba con un caso de prueba nuevo que *mate* al *mutante*.

Con los anteriores conceptos ya definidos estamos en disposición de decir que el desarrollo del PFC se divide en 2 fases: una primera, de creación de una composición en lenguaje WS-BPEL 2.0 a la que se le puedan aplicar el mayor número de operadores de mutación posibles y otra, en la que se busca un conjunto de casos de prueba de buena calidad.

La temática de la composición WS-BPEL 2.0 fue también propuesta por el propio grupo y ésta trata de modelar la concesión de un préstamo personal por parte de una

entidad. En un primer instante la idea era ampliar una versión previa ya existente pero finalmente se decidió crear una nueva desde el inicio.

El proyecto se ha elaborado en 9 meses, desde noviembre de 2009 hasta julio de 2010 y se ha dividido en las 8 fases siguientes:

- Aprendizaje de la especificación WS-BPEL 2.0
- Estudio de los operadores de mutación.
- Diseño general de la estructura de la composición.
- Información sobre préstamos personales.
- Familiarización con el IDE Netbeans - Inicio del desarrollo básico de LoanApprovalExtended.
- Desarrollo de LoanApprovalExtended.
- Refinamiento de la composición. Fase investigación.
- Desarrollo de la memoria.

Las etapas más laboriosas fueron la de aprendizaje de la especificación, desarrollo de la composición y la fase de investigación.

En cuanto a la composición, el enfoque que se le ha dado a esta composición es la de un trabajador de una entidad bancaria el cual tiene que responder a un cliente sobre la aprobación o denegación de un préstamo. El cliente desea saber si le concederán el préstamo y las condiciones de reembolso del mismo, por lo cual se le debe contestar con un sí y la cuota mensual que deberá pagar durante el periodo de reembolso o el no con la razón de la denegación.

Las condiciones de denegación son las siguientes. El cliente:

- Es moroso.
- Tiene una propiedad embargada.
- Está actualmente en paro.
- No supera el canon francés.
- Ha trabajado poco a lo largo de su vida. Esto ocurre cuando una persona ha trabajado muy pocos días en relación al número de días que podría haber trabajado desde que tuvo algún empleo por primera vez. Atendiendo a su pasado podemos deducir que en caso de pérdida del empleo no podrá encontrar uno nuevo con facilidad.
- Relación *cantidad solicitada / meses devolución*. Debe existir una concordancia entre la cantidad solicitada y el número de meses en los que se desea realizar el reembolso.

- No supera una evaluación global (Se realiza una evaluación global que reúne a las anteriores y algunos factores más como situación personal, estado civil, profesión, etc).

El funcionamiento de la composición es el siguiente:
Recibe como entrada varios datos del cliente como por ejemplo: cantidad solicitada, número de meses en los que se desea reembolsarla, gastos mensuales, nivel de estudios, estado civil, etc. Nada mas empezar la ejecución evaluamos el cociente entre los meses en los que se desea reembolsar el préstamo y la cantidad solicitada. A continuación se ejecutan las llamadas a los cuatro servicios Web (*mockups*, ya que son Servicios Web simulados) que nos aportarían la información restante del cliente.

Estos cuatro son los siguientes:

- Seguridad Social: Proporciona la vida laboral del cliente. Evaluaremos su situación laboral actual y pasada.
- Banco de España: Conseguimos la CIRBE donde vemos los préstamos e hipotecas que posea el cliente.
- Ministerio de Justicia: Obtenemos el Registro de la Propiedad para saber si tiene alguna propiedad embargada.
- ASNEF: Nos dice si el cliente es moroso.

En base a toda esta información obtenida se evalúan los datos y se dictamina si se concede o no el préstamo. En caso de ser necesario también se puede considerar la figura del aval para que asegure toda la operación.

El perfil del usuario de este PFC principalmente son los miembros de SPI&FM ya que son los que lo han promovido. No obstante también puede estar encaminado a algún estudiante que desee realizar un PFC similar o alguien que quiera desarrollar alguna composición en WS-BPEL y necesite orientación básica acerca del lenguaje y todo lo que le rodea. Es por ese motivo por el que se han realizado una serie de apéndices que pueden servir de gran ayuda. Éstos son:

- WS-BPEL 2.0: En este apartado se introducen brevemente los conceptos más importantes correspondientes a WS-BPEL 2.0. Será de gran utilidad puesto que no existen muchos documentos sobre WS-BPEL 2.0 en español.
- Operadores de mutación: Se explican cuáles son los operadores existentes y cómo funcionan.
- Algunos errores al programar con WS-PBEL en GAmera: Se dan a conocer algunos de los errores más cometidos en el desarrollo de la composición.
- De NetBeans a GAmera: Apartado de grandísima utilidad en el que se explica el proceso de creación de un proyecto en NetBeans además de cómo se debe modificar y qué ficheros crear para que esa misma composición sea ejecutable en GAmera.

En la elaboración del proyecto se han adquirido los siguientes conceptos:

- Aprendizaje de las siguientes tecnologías:
  - WS-BPEL 2.0: Con el cual vamos a crear la composición.
  - WSDL: Con el que crearemos las interfaces de interacción entre la composición y el exterior.
  - XML Schema: Con el que podremos crear los tipos de datos.
  - XPath: Haremos uso de él en la composición para crear expresiones de consulta.
  - LaTeX: Utilizado para desarrollar este documento.
  - SVN: Control de versiones para desarrollar la composición de manera segura.
- Uso del framework de pruebas BPELUnit para ejecutar "localmente" la composición.
- Mutación, mutantes, operadores de mutación, estado de los mutantes.
- Importancia de herramientas de verificación de software que nos permitan localizar fallos en el proceso de programación que en un principio no son fáciles de hallar.
- Conceptos relacionados con los préstamos personales y el proceso lógico que se debe seguir en la concesión de un préstamo de esas características.
- Manejo de entornos de desarrollo integrados que facilitan la programación en WS-BPEL como son NetBeans, ActiveVOSDesigner y Eclipse.
- Interpretación y análisis de los resultados obtenidos tras ejecutar y procesar con GAmera la composición.

Para el desarrollo del proyecto se ha utilizado el modelo en incremental obteniendo versiones cada vez más complejas y elaboradas de la composición, ampliando progresivamente el número de operadores de mutación utilizados.

En cuanto a la recolección de requisitos, no fueron necesarias muchas reuniones con el grupo ya que las ideas estaban muy claras desde un principio (crear una composición de gran tamaño a la que se le pudiera aplicar el mayor número de operadores de mutación diferentes posibles) y dieron total libertad para realizar la composición.

Si nos centramos en los datos obtenidos de la ejecución y análisis de la composición nos encontramos con:

- Número de mutantes generados: 3374.
- Porcentaje de mutantes válidos: 100 %.
- Porcentaje de mutantes muertos: 67,93 %.
- Porcentaje de mutantes repetidos: 0 %.
- Número de casos de prueba: 40.

6 Conclusiones

Las conclusiones que se pueden obtener tras este PFC son las siguientes:

- Hemos obtenido conocimientos sobre las mutaciones, el comportamiento y funcionamiento de los operadores mutantes.
- Aplicamos el método incremental para la elaboración del proyecto. También hemos estudiado versiones anteriores de composiciones que simulan la concesión de préstamos personales para observar y ampliar con posibles mejoras. Para la creación del PFC hemos ido creando versiones y prototipos que aumentaban en complejidad y tamaño.
- Aprendimos las siguientes tecnologías implicadas en el proyecto: WS-BPEL, WSDL, XPath, XML Schema.
- Demostración empírica durante el desarrollo de la composición de que GAmera permite depurar un código WS-BPEL en busca de errores de programación que no son apreciables con facilidad.
- La necesidad de la existencia de una herramienta como GAmera que permita depurar código WS-BPEL 2.0.
- Aprendizaje autodidacta en la mayor parte del proyecto pero con la ayuda imprescindible de manera puntual por parte del tutor.

LoanApprovalExtended es una composición que amplía y mejora los resultados que se habían obtenido con la versión previa existente.

El estudio de los lenguajes y las IDE de desarrollo han resultado la parte más dura del proyecto junto con BpelUnit por lo que se valora más que nunca en estas situaciones la figura del docente y las clases impartidas por los mismos. Respecto a la segunda fase del proyecto no resultó muy complicada aunque sí se hizo eterna ya que es un proceso que requiere mucho tiempo y paciencia.

7 Manual de instalación

7.1. Instalación de GAmera

Existen dos métodos para instalar GAmera en nuestro equipo:

- Haciendo uso de *subversion*.
- Guión de instalación a través de un cliente *wget*.

7.1.1. A través de subversion

Éste es bastante sencillo ya que para la instalación de GAmera existe un *script* el cual la realiza automáticamente casi en su totalidad. Para instalar GAmera en nuestra máquina debemos previamente hacernos con una copia de los fuentes. Éstos los obtenemos de la forja Redmine que está alojada en el servidor Neptuno del Departamento de Lenguajes y Sistemas Informáticos. Para conseguirla debemos tener instalado previamente *subversion*.

Subversion es un software de sistema de control de versiones. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como *svn* por ser ese el nombre de la herramienta utilizada en la línea de órdenes.

Si es necesario instalar *subversion*, debemos ejecutar las siguientes órdenes en consola:

1. *sudo update*
2. *sudo upgrade*
3. *sudo apt-get install subversion*

Con *subversion* en nuestra máquina escribimos la siguiente orden en la consola:
svn co https://neptuno.uca.es/svn/sources-fm/trunk

Con esta orden obtendremos una copia del repositorio con la que trabajar. En ella obtendremos todos los fuentes de las herramientas que serán necesarias para ejecutar correctamente GAmera. La copia será creada en el directorio que hayamos ejecutado la orden. Una vez con la copia de trabajo, nos dirigimos al directorio *scripts* y en consola ejecutamos el *script* de instalación de GAmera:
./install.sh gamera

La instalación puede durar bastante tiempo así que hay que tener paciencia. Una vez GAmera ha sido instalada las siguientes variables de entorno se deberían establecer automáticamente. En caso de que no fuese así debemos establecer las variables de entorno necesarias. Estas son:

- *JAVA_HOME,JDK_HOME*
- *CATALINA_HOME*

Con los siguientes valores:

- *JAVA_HOME=/usr/lib/jvm/java-6-sun*
- *JDK_HOME=/usr/lib/jvm/java-6-sun*
- *CATALINA_HOME= /bin/tomcat5*

Probablemente para que los cambios surtan efecto convendría salir de la sesión de usuario y volver a entrar.

7.1.2. Guión de instalación mediante wget

Podemos obtener una copia del trabajo mediante una simple consulta HTTP, con el cliente *wget*, que viene de serie prácticamente en todas las distribuciones de hoy en día. La consulta HTTP es la siguiente:

wget https://neptuno.uca.es/svn/sources-fm/trunk/scripts/install.sh

El guión establecerá las variables de entorno por sí mismo. De igual modo convendría salir de la sesión de usuario y volver a entrar.

8 Manual de usuario

8.1. Manual de usuario de GAmera

En esta sección se describirá un pequeño manual para utilizar GAmera. Para ser exactos no usaremos GAmera propiamente dicho sino *mutationtool*, herramienta de mutación ya introducida en otros apartados de este documento. Se describen los pasos que debemos seguir para trabajar con la composición atendiendo a las opciones que ofrece *mutationtool*. Se explicarán sólo las que más se han usado para desarrollar este PFC.

Si deseamos ejecutar la composición debemos tener arrancado el motor *ActiveBpel*. Para ello introducimos en la consola:

ActiveBpel start

Una vez arrancado, situándonos en el directorio de la composición comprobamos que tenemos los ficheros necesarios para la ejecución. Éstos son:

- *LoanApprovalExtended.bpel*
- Los diferentes archivos *.wsdl* y *xsd*
- *LoanApprovalExtended.bpts*, en el que tenemos los casos de pruebas que deseamos ejecutar.

En estas condiciones podemos trabajar con la composición.

Manual mutationtool

Si introducimos el siguiente comando en consola:

mutationtool

Nos aparece el menú de opciones que ofrece mutationtool:

- analyze bpel
- apply bpel operator operand attribute

- applyall bpel
- run bpts bpel
- compare bpts bpel xml bpel1...
- comparefull bpts bpel xml bpel1...
- compareout xml1 xml2
- normalize bpel

A continuación se explica brevemente qué realiza alguna de sus opciones:

8.1.1. mutationtool analyze .bpel

mutationtool analyze LoanApprovalExtended.bpel

Con esta opción obtendremos un listado en el que podremos observar los operadores de mutación existentes junto a dos números. El primer número, indica el número de veces que podremos aplicar dicho operador de mutación a la composición y el segundo, indica el número de atributos que posee dicho operador. El número de mutantes finales que podremos obtener para el operador en cuestión vendrá dado por el producto de estas dos cifras.

Por ejemplo:

ISV 1684 1
EAA 51 4
EEU 6 1
ERR 95 5
ELL 20 1
ECC 103 1
ECN 137 4
EMD 0 2
EMF 0 2
ACI 0 1
AFP 2 1
ASF 45 1
AIS 2 1
AIE 55 1
AWR 4 1
AJC 1 1
ASI 159 1
APM 0 1
APA 0 1
XMF 24 1
XMC 1 1
XMT 0 1

XTF 32 1
XER 10 1
XEE 0 1

8.1.2. mutationtool apply bpel operator operand attribute

Crearemos un mutante para el operador *operator*. Hay que especificarle el operando y el atributo con el cual queremos crearlo. Por ejemplo: *mutationtool apply LoanApprovalExtended.bpel 2 40 2*

Con esa combinación indicamos:

- Operador de mutación EAA , que es el segundo.
- De las 51 posibilidades en las que se puede aplicar el operador, elegimos la número 40.
- De los 4 atributos que tiene ese operador, nos quedamos con la número 2.

El resultado es el siguiente. En primer lugar tenemos el trozo de código original:

```
<if>
        <condition>$_diaVieja &lt;= 27</condition>
        <sequence>
        <assign>
                <copy>
                <from>$_diaVieja + 1</from>
                <to variable="_diaVieja"/>
                </copy>
        </assign>
        </sequence>
<else>
```

Y a continuación el mutante generado:

```
<if>
        <condition>$_diaVieja &lt;= 27</condition>
        <sequence>
        <assign>
                <copy>
                <from>$_diaVieja * 1.0</from>
                <to variable="_diaVieja"/>
                </copy>
        </assign>
        </sequence>
<else>
```

Observamos que la mutación producida consiste en la sustitución del operador de suma por el operador de multiplicación.

8.1.3. mutationtool applyall bpel

Applyall nos generará en el directorio actual todos los mutantes posibles para la composición. La orden *mutationtool applyall LoanApprovalExtended.bpel* genera 3374 mutantes diferentes.

8.1.4. mutationtool run bpts bpel

Permite ejecutar la composición y observar la salida de la misma en la salida estándar. Si se quiere obtener en un fichero, la orden debería ser: *run bpts bpel >fichero.xml*

mutationtool run LoanApprovalExtended.bpts LoanApprovalExtended.bpel >ficheroSalida.xml

No se muestra *ficheroSalida.xml* ya que su longitud es demasiada como para mostrarla en este documento (17000 líneas aproximadamente). No obstante, se describe parte del fichero.

El visionado y comprensión de la salida no es sencilla así que es recomendable utilizar *XMLEye*, un programa desarrollado por Antonio García Domínguez y con el cual podremos otear los resultados de manera sencilla.

A continuación hay una breve explicación de la estructura principal de la salida para comprender los resultados de la ejecución:

Para cada uno de los casos de prueba tenemos:

```
<tes:testCase name="Test Case testCase0-b-Concedido"
result="PASSED" message="Passed" >
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
</tes:testCase>
```

En su interior tenemos:

Un bloque que es el encargado de comunicarse con el cliente.

```
<tes:partnerTrack name="Partner Track client" result="PASSED"
message="Passed" >
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
...
</tes:partnerTrack>
```

Una serie de bloques que representan a los Servicios Web.

```
<tes:partnerTrack name="Partner Track PL_SSOCIAL"
result="PASSED" message="Passed" >
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
...
</tes:partnerTrack>
```

Cada uno de estos bloques en su interior tienen una actividad.

```
<tes:activity type="ReceiveSendSync" name="Receive/Send Synchronous"
result="PASSED" message="Passed" >
        <tes:state name="Status Code">PASSED</tes:state>
        <tes:state name="Status Message">Passed</tes:state>
..
</tes:activity>
```

Y dentro de las actividades, tenemos los mensajes:
En un sentido.

```
<tes:dataPackage name="Send Data Package" result="PASSED"
message="Passed">
        ...
        <tes:state name="Direction">INPUT</tes:state>
        ...
                <literalData>
                        <ns1:datos ...>
                                <ns0:dni>'31709567L'</ns0:dni>
                                <ns0:cantidad>6000</ns0:cantidad>
                                <ns0:numMesesPagar>12</ns0:
                                   numMesesPagar>
                                ...
                        </ns1:datos>
                </literalData>
</tes:dataPackage>
```

Y en el contrario.

```
<tes:dataPackage name="Receive Data Package" result="PASSED"
message="Passed">
        ...
        <tes:state name="Direction">OUTPUT</tes:state>
        ...
                        <soapenv:Body>
                                <cuota>521.930574512531</cuota>
                                <dictamen>true</dictamen>
                                <razon>Cumple todos los requisitos</
                                   razon>
                        </soapenv:Body>
        ...
</tes:dataPackage>
```

Cuando ejecutamos cada uno de los casos de prueba el resultado que podemos obtener es el siguiente:

- Concedido, cuando el cliente cumple todos los requisitos necesarios para obtener el préstamo.
- Denegado, cuando el cliente no cumple al menos uno de los requisitos necesarios.
- Error, si se ha producido algún error en la ejecución de la composición.

Las razones de denegación contempladas en torno al cliente han sido las siguientes:

- Es moroso.
- Tiene una propiedad embargada.
- Está actualmente en paro.
- No supera el canon francés.
- Ha trabajado poco a lo largo de su vida. Esto ocurre cuando una persona ha trabajado muy pocos días en relación al número de días que podría haber trabajado desde que tuvo algún empleo por primera vez. Atendiendo a su pasado podemos deducir que en caso de pérdida del empleo no podrá encontrar uno nuevo con facilidad.
- Relación *cantidad solicitada / meses devolución*. Debe existir una concordancia entre la cantidad solicitada y el número de meses en los que se desea realizar el reembolso.
- No supera una evaluación global. Se realiza una evaluación global que reúne a las anteriores y algunos factores más como situación personal, estado civil, profesión...

Las situaciones de error contempladas son las siguientes:

- Fallo en la vida laboral.
- Error en el nivel de estudios.
- Estado de la propiedad erróneo.
- Grupo de profesión erróneo.
- Estado civil erróneo.
- Cantidad solicitada fuera de los límites.
- Numero de meses en la devolución erróneo.

8.1.5. mutationtool compare bpts bpel xml bpel1...

Compare nos creará una matriz de *1xN* (siendo *N* el número de casos de prueba del fichero *bpts*) en el que nos aparecerán una fila de ceros (para los casos de prueba que no maten al mutante) y un uno (en el que caso que algún caso de prueba mate al mutante). Tan sólo aparecerá un uno ya que una vez que encuentra al primer caso de prueba que mata al mutante (y eso es lo que importa) el resto no es tenido en cuenta y se pone a cero. Esto se hace para disminuir el tiempo de ejecución y ahorrar recursos.

También podemos volcar la salida en un fichero del siguiente modo:

mutationtool compare LoanApprovalExtended.bpts LoanApprovalExtended.bpel ficheroSalida.xml m24.bpel >salidaXMF.out*

El contenido de *salidaXMF.out* está descrito a continuación: Tenemos una matriz de M_{24x22}, donde 24 es el número de mutantes que existe para el operador XMF y 22 el número de casos de prueba que tenemos en *LoanApprovalExtended.bpts*. El valor que podemos encontrar en M_{ij} es cero o uno. Uno si el caso de prueba *j* mata al mutante *i* y cero si no lo hace.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ->Mutante Vivo
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ->Mutante Vivo
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ->Mutante Muerto
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

8.1.6. mutationtool comparefull bpts bpel xml bpel1...

Esta opción es más completa que la anterior y comprueba si todos los casos de prueba matan al mutante a pesar de que ya haya encontrado a otro anteriormente que lo haya matado.

mutationtool comparefull LoanApprovalExtended.bpts LoanApprovalExtended.bpel ficheroSalida.xml m13.bpel >salidaCompareFullAIS.txt*

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
Mutante Muerto. Matado por dos casos de prueba.

8.1.7. mutationtool normalize bpel

Esta opción permite obtener en la salida estándar una versión normalizada del fichero *bpel* original. Ésta estará convenientemente identada y estructurada.

A WS-BPEL 2.0

A.1. Introducción

A.1.1. Terminología

WS-BPEL es el acrónimo para Web Services Business Process Execution Language. WS-BPEL 2.0 es una revisión de la especificación BPEL4WS (Business Process Execution Language for Web Services) 1.0 y 1.1.

A.1.2. Historia de WS-BPEL

XLANG y WSFL

Business Process Execution Language for Web Services (BPEL4WS) fue concebido por primera vez en julio de 2002 con el lanzamiento de la especificación BPEL4WS 1.0, un esfuerzo conjunto de IBM, Microsoft y BEA. Este documento propuso un lenguaje de orquestación inspirado en variaciones previas, como eran *Web Service Flow Language* (WSFL) de IBM y la especificación XLANG de Microsoft.

Un año más tarde, en mayo de 2003 apareció la especificación BPEL4WS 1.1, esta vez con ayuda de SAP y Siebel Systems. Esta versión recibió mas atención y ayuda por parte de los fabricantes, dando lugar a un número de motores de orquestación para BPEL4WS disponibles comercialmente. Antes del lanzamiento, la especificación BPEL4WS fue sometida a un comité técnico de OASIS con la finalidad de que la especificación pudiera ser desarrollada y convertirse en un estándar abierto.

A.2. Conceptos básicos

A.2.1. Estructura de los Procesos BPEL

En primer lugar, mencionar que un Proceso BPEL es un contenedor donde puedes declarar relaciones hacia socios externos, declaraciones de Datos de Proceso, manejadores con varios propósitos y actividades a ser ejecutadas.

En la parte superior de una composición BPEL, encontramos el contenedor *process*. Este contenedor tiene un par de atributos que son:

- nombre(*name*)
- *xmlns*. Declaración de un espacio de nombres xml.

```
<process name="Process"
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
executable"/>
```

La declaración de espacio de nombres *http://docs.oasis-open.org/wsbpel/2.0/process/executable* indica que se trata de un proceso ejecutable. Existe otro espacio de nombres para procesos abstractos. La diferencia entre ambos es que en los ejecutables se definen las dos partes que componen el comportamiento de un negocio (lo externamente visible y la parte de procesamiento interna) y en un abstracto no.

El elemento *process* es el contenedor más exterior. Por tanto, cualquier socio, dato de proceso o manejador que se declaren dentro del *process* pueden ser considerados como globales. Si queremos hacer cualquier tipo de declaración local, en BPEL existe el elemento llamado *scope*.

```
<scope name="Scope" />
```

A.2.2. Relaciones con socios de negocio

Los procesos de negocio de BPEL ofrecen la posibilidad de agregar servicios web y definir la lógica de negocio entre cada uno de esos servicios, es por ello que se dice que BPEL es un lenguaje de orquestación y orquesta tales interacciones de servicios Web. Cada interacción de servicio puede ser vista como una comunicación con un socio de negocio. Esa interacción está descrita con ayuda de enlaces a socios de negocio (*partner links*). Los *Partner Links* son instancias de unos conectores que especifican los *port types* de WSDL que el proceso ofrece y requiere desde un socio, al otro lado del *partner link*.

Por cada socio, se puede tener un conjunto de *partner links*. Cada interacción tiene 2 partes o caras: El proceso invoca al socio y éste hace lo propio con el primero. Cada *partnerLink* está caracterizado por un *partner link type* y un nombre de *role* que describe la función dentro del enlace.

```
<partnerLinks>
<partnerLink name="Client" partnerLinkType="wsdl:ClientPLT"
 myRole="Client" />
</partnerLinks>
```

Las declaraciones de *partnerLinks* pueden estar situadas directamente bajo un elemento *process*, lo cual significa que son accesibles por todos los constructores dentro del proceso BPEL o bajo un elemento *scope*, lo cual significa que son accesibles únicamente desde los hijos de ese mismo *scope*.

A.2.3. Estado de un Proceso BPEL

Anteriormente se introdujo el concepto de Dato de Proceso. Técnicamente, un dato de proceso son variables que son declaradas en un *process* o en un *scope*. Las variables contienen el dato que constituye el estado de un proceso de negocio BPEL durante la ejecución. El valor de las variables proviene de 2 fuentes: de mensajes intercambiados con un socio o es el resultado datos intermedios que son privados al proceso.

Tipos de variables existentes en BPEL:

- WSDL *message types*
- XML Schema *simple types*
- XML Schema *complex types*
- XML Schema *elements*

Puedes declarar una variable especificando un nombre de uno de los tipos mencionados anteriormente escribiendo de una de las tres maneras que se ven a continuación: *type, messageType o element*.

```
<variables>
<variable name="V1" messageType="NS:WSDLMessageDataType"  />
<variable name="V1" element="NS:XMLElement" />
<variable name="V2" type="xsd:boolean" />
<variable name="V2" type="NS:ComplexType" />
</variables>
```

A.2.4. Comportamiento de un Proceso BPEL

Un Proceso BPEL está constituido principalmente por actividades las cuales pueden ser de dos tipos:

- Estructurada, las cuales contienen a otras actividad y la lógica de negocios entre ellas.
- Básicas, que desarrollan el cometido para el que fue creado como es recibir mensajes, manipular datos ...

Proporcionar y consumir Servicios Web

En BPEL existen 3 actividades simples para consumir mensajes desde y proporcionar mensajes a servicios Web socios. Éstas son *receive*, *reply* e *invoke*. Estas tres actividades permiten intercambiar mensajes con socios externos.

El propósito de *receive* es recibir mensajes desde socios externos. En ella siempre se debe especificar un *partnerLink*, una operación (*operation*) y una variable donde almacenar el dato recibido.

```
<receive name="Receive" createInstance="yes"
partnerLink="ClientPLT" operation="Op1".../>
```

Con el atributo *createInstance="YES"* se crea una nueva instancia del Proceso. En caso de tener el valor *"NO"* el mensaje entrante, será consumido por la instancia del proceso existente.

Cada actividad *receive* puede tener asociada una actividad *reply*. La actividad *reply* puede devolver dos tipos de datos: un dato normal (*normal data*) y un dato de error (*faulted data*). En este último caso se debe especificar un atributo *faultName*. A esta actividad también se le debe indicar el nombre de una operación y un *partnerLink*.

```
<reply name="Reply" partnerLink="ClientPLT" operation="Op1"
... />
```

La última actividad que nos queda es *invoke*. Se usa para llamar a un servicio proporcionado por un socio externo. Se especifica un *partnerLink* y una operación.

```
<invoke name="InvokeWS" partnerLink="BusinessPartnerSL"
operation="partnerOp" ... />
```

En WSDL existen varios tipos de operaciones. Dos de ellas son soportadas por BPEL: operaciones *one-way* (se puede continuar con la lógica del proceso sin esperar al socio para efectuar el *reply*) y operaciones *request-response* (en las que se bloquea el proceso o parte de él hasta recibir la respuesta por parte del socio). Una actividad *invoke* soporta a ambas. Si la operación es del segundo tipo se debe especificar una variable de entrada y otra de salida.

```
<invoke name="ReqRespInv" partnerLink="BusinPartSL"
operation="ReqRespOp" inputVariable="In"
outputVariable="Out" />
```

Si es una operación *one-way* sólo se especifica una variable de entrada.

```
<invoke name="1WayInv" partnerLink="BusinPartSL"
 operation="1WayOp" inputVariable="Input" />
```

Estructurando al proceso lógico

BPEL permite estructurar la lógica de negocio de acuerdo a tus necesidades. En el caso de necesitar ejecutar un número de actividades de manera secuencial, ésto se puede hacer usando la actividad *sequence*. En otras palabras, la actividad *sequence* es usada para definir una colección de actividades la cuales serán ejecutadas de forma secuencial en orden léxico.

```
<sequence name="Sequence">
<receive name="receiver" ... />
<invoke name="checkt" ... />
<invoke name="shipping" ... />
<reply name="send" ... />
</sequence>
```

Otra actividad estructural es *if-else* la cual permite seleccionar exactamente una de las diferentes ramas de las que se compone esta actividad. Para cada rama, se debe chequear una condición y en el caso de que sea evaluada como *true*, entrará en dicha rama. Como todas las expresiones en BPEL, se pueden usar expresiones XPath para formular la condición. Si ninguna condición es evaluada como *true*, entonces una elección por defecto puede ser especificada usando la rama *else*.

```
<if name="ifname">
        <condition>
                $var1 &gt; 5000
        </condition>
        <invoke name="calc1" ... />
<elseif>
        <condition>
                $var2 &gt; 2500
        </condition>
        <invoke name="calc2" ... />
</elseif>
<else>
        <reply name="send" ... />
</else>
</if>
```

Actividades repetitivas

BPEL ofrece tres actividades que permiten la ejecución repetida de un trozo de lógica de negocio. Una de ellas es *while*. Esta actividad debe tener al menos una actividad hija en su interior. *While* permite la ejecución repetida de las actividades hijas siempre y cuando la condición sea evaluada como *true*. Esta condición se evalúa al comienzo de la iteración.

```
<while>
<condition>
        $iterations &gt; 5
</condition>
        <invoke name="interator"... />
</while>
```

Por contra existe otra actividad repetitiva, *repeatUntil* que tiene como diferencia en su comportamiento el hecho de que su cuerpo (actividades hijas) se ejecutan al menos una vez, ya que la condición que evalúa la repetición se realiza al final de cada iteración.

```
<repeatUntil>
        <invoke name="iterator" ... />
<condition>
        $iterations &gt; 5
</condition>
</repeatUntil>
```

La tercera actividad repetitiva es *ForEach*. En su comportamiento por defecto, esta actividad itera de forma secuencial *N* veces sobre un conjunto de actividades dadas. *N* es igual a *finalCounterValue* menos *startCounterValue + 1*.

```
<forEach parallel="no" counterName="N" ...>
<startCounterValue>1</startCounterValue>
<finalCounterValue>10</finalCounterValue>
<scope> ...</scope>
</forEach>
```

Hay dos variantes de la actividad *ForEach*: secuencial y paralelo. Ésto es especificado por el atributo *parallel* que se detallará más adelante.

Procesado paralelo

En caso de ser necesario ejecutar diferentes actividades de manera paralela, BPEL proporciona la actividad *flow*. Además de esta actividad se puede obtener procesado

en paralelo con la variante *parallel* de la actividad *ForEach* y también con el uso de manejadores de eventos.

```
<flow ...>
<invoke name="comprobarVuelo" ... />
<invoke name="comprobarHotel" ... />
<invoke name="comprobarAlquierCoche" ... />
</flow>
```

Si dentro de una actividad *flow* queremos sincronizar actividades, haremos uso de *links*, *sources* y *targets*. Cuando una actividad tiene asociado un *target*, esa actividad se ejecutará si y solo si, otra actividad que tendrá asociado un *source* ha finalizado su ejecución. Estas actividades se hacen corresponder las unas con las otras a través de *links*.

```
<flow ...>
<links>
<link name="checkFlight-To-BookFlight" />
</links>
<invoke name="checkFlight" ...>
        <sources>
        <source linkName="checkFlight-To-BookFlight" />
        </sources>
</invoke>
<invoke name="checkHotel" ... />
<invoke name="checkRentalCar" ... />
<invoke name="bookFlight" ...>
        <targets>
        <target linkName="checkFlight-To-BookFlight" />
        </targets>
</invoke>
</flow>
```

En el ejemplo anterior, la actividad <*invoke name*="*bookFlight*"...> se ejecutará únicamente si <*invoke name*=*çheckFlight*"...> ha finalizado su ejecución.

Manipulación de datos

En ocasiones es necesario que los datos que vamos a enviar/recibir sean troceados en varias partes, o unidos desde varias fuentes.

BPEL tiene una actividad llamada *assign* la cual contiene una o más operaciones *copy* en su interior y a su vez cada *copy* contiene un *from* y un to indicando la fuente y destino desde donde han sido obtenidos los datos o hacia donde se enviarán.

```
<assign>
<copy>
<from variable="var1" />
<to variable="var2" />
</copy>
</assign>
```

Si queremos copiar partes de una variable a otra variables se puede realizar de la siguiente manera.

```
<assign>
<copy>
<from variable="Input"
part="operation1parte/myNS:InformacionTarjetaCredito"/>
<to variable="EntradaServicioTarjetaCredito" />
</copy>
</assign>
```

Manejo de Excepciones

Hasta ahora hemos dado por hecho que todas las ejecuciones se realizan correctamente y sin problemas pero un lenguaje como BPEL debe estar preparado para manejar situaciones imprevistas como por ejemplo, llamar a un servicio Web y que éste no esté disponible. Para estos casos BPEL posee el concepto de *fault handlers*. Un *fault handler* puede estar enlazado a un *scope*, un *process*, o un *invoke*. Con *catch faultName* capturamos un fallo determinado y se ejecutan las operaciones hijas que se encuentran dentro de él.

```
<faultHandlers>
<catch faultName="ExcepcionSinStock"
faultVariable="VariableSinStock">
...
</catch>
<catchAll>...</catchAll>
</faultHandlers>
```

Catch posee dos atributos, *faultName* y *faultVariable*. Cuando éste último es usado, o bien *faultMessageType* o bien *faultElement* deben ser especificados. *FaultVariable* apuntaría a una variable de ámbito local dentro de del *catch* siendo ésta de los tipo *faultMessageType* o *faultElement*.

A.3. Conceptos avanzados

A.3.1. Refinamiento de la estructura del Proceso

En BPEL, es posible estructurar tus procesos de negocios en una jerarquía de scope anidados. Cada scope puede tener sus propios definición de variables, *partnerlinks*, intercambio de mensajes, *correlation sets*, y manejadores. Esto limita la visibilidad de estas definiciones a las actividades adjuntas y establece el contexto en el cual ellas son ejecutadas.

Dos constructs de BPEL requieren el uso de scopes:

- La actividad primaria en el manejador de eventos *onEvent*.
- La actividad *forEach*.

Manejo de Fallos en un Scope

Un *faultHandler* puede estar asociado a un *scope* con el fin de hacer frente a situaciones excepcionales de forma más localizada en el lugar en el que han ocurrido.

```
<scope>
<faultHandlers>
        <catch faultName="myNS:ErrorInesperado">...</catch>
        <catchAll><!-- manejamos otros errores -->

        </catchAll>
</faultHandlers>
<sequence>

</sequence>
</scope>
```

Deshacer trabajo ya finalizado

Los procesos de negocio representan normalmente trabajo de larga duración el cual no puede ser finalizado dentro de una transacción “simple atómica”. Las transacciones ACID ya realizadas crean efectos persistentes antes de que el proceso esté completo del todo. BPEL proporciona un mecanismo por el cual se realizan unas medidas de compensación que deshacen esos efectos cuando sea requerido. El mecanismo se llama *compensationHandler* el cual es invocado por las actividades *compensate* y *compensateScope*. Este mecanismo funciona correctamente si y sólo si ha finalizado de manera satisfactoria el *scope* al cual está asociado.

```
<scope name="Scope1">
<faultHandlers>
<catchAll>
<compensateScope target="Scope2" />
</catchAll>
</faultHandlers>
<sequence>
<scope name="S2">
<compensationHandler>
<!--deshacer trabajo -->
</compensationHandler>
<!-- hacer algun trabajo -->
</scope>
<!--realiza mas trabajo -->
<!-- lanzamos una falta, el trabajo de Scope2
debe deshacerse -->
</sequence>
</scope>
```

Compensate y *compensateScope* están situados en *faultHandlers*, *compensationHandler* y *terminationHandlers*. Éstos últimos se encuentran inmediatamente adjuntados a un *scope*.

A.3.2. Interacciones avanzadas con Servicios Web

BPEL posee diferentes constructs para modelar cada una de las situaciones que se pueden dar en la recepción de un mensaje.

Procesamiento selectivo de eventos

En el primer escenario complejo imaginemos la siguiente situación. Nos encontramos con un conjunto de mensajes que se recibirán en un proceso y donde cada uno de ellos puede provocar una serie de pasos posteriores en el proceso de negocio. Tan sólo uno de ellos realiza esos pasos finalmente. Opcionalmente se permite especificar el comportamiento que deberá tener el proceso para el caso excepcional donde ninguno de esos mensajes entrantes llegan dentro de una cantidad de tiempo determinada. BPEL modela esta situación con la actividad *pick*. En ella, tenemos una serie de actividades *onMessage* y 0 o más elementos *onAlarm*. Por cada *onMessage* se espera un mensaje entrante. Dentro de dicha actividad, tendremos un conjunto de actividades hijas que se ejecutarán cuando llegue el mensaje (estos son los pasos posteriores a los que nos referíamos anteriormente). Cada *onAlarm* tiene un *point in time* específico o un intervalo

de tiempo máximo para que llegue alguno de los mensajes esperados. En caso de que no lleguen en ese intervalo, se ejecutarán las actividades hijas del *onAlarm*.

```
<pick>
<onMessage partnerLink="buyer" operation="input"
variable="Item">
<!--realizar trabajo 1 -->
</onMessage>
<onMessage partnerLink="buyer" operation="order"
variable="Detail">
<!--realizar trabajo 2-->
</onMessage>
<onAlarm>
<for>'P10DT3H'</for>
<!--realizar trabajo 3-->
</onAlarm>
</pick>
```

Procesamiento múltiple de eventos

En el caso anterior un mensaje provocaba una serie de pasos posteriores. En este escenario tenemos que más de un mensaje son necesarios para provocar esos pasos.

```
<flow>
<links>
        <link name="Comprador" />
        <link name="Vendedor" />
</links>
<receive name="InformacionDesdeComprador"
createInstance="yes" ...>
        <sources>
        <source linkName="Comprador" />
        </sources>
        <correlations>
        <correlation set="tradeID" initiate="join" />
        </correlations>
</receive>
<receive name="InformacionDesdeVendedor"
createInstance="yes" ...>
        <sources>
        <source linkName="Vendedor" />
        </sources>
        <correlations>
        <correlation set="tradeID" initiate="join" />
        </correlations>
```

```
</receive>
<invoke name="Comerciar" ...>
        <targets>
        <joinCondition>$Comprador and $Vendedor</joinCondition>
        <target linkName="Comprador" />
        <target linkName="Vendedor" />
        </targets>
</invoke> ...
</flow>
```

En este caso se pueden recibir más de un mensaje pero no al mismo tiempo. Deben llegar unos antes que otros. Una vez que hayamos recibido información de parte del comprador y de parte del vendedor(no al mismo tiempo) podremos ejecutar el *invoke* que se encarga de comerciar. Con *jointCondition* podemos crear una barrera la cual no puede sobrepasarse hasta que los sources correspondientes (Comprador y Vendedor) no hayan finalizado.

Procesamiento concurrente de eventos

En este caso se recibe más de un mensaje y esta vez pueden recibirse al mismo tiempo. El proceso de negocio está de esta manera preparado para aceptar peticiones que llegan en paralelo. En BPEL esto lo conseguimos con la actividad *EventHandling*. En su interior encontramos actividades *onEvent* que actúan de manera similar al *onMessage* anteriormente mencionado.

```
<process name="Process" ...>
<eventHandlers>
        <onEvent partnerLink="pl1"
operation="ConsultarEstatusPeticion" ...>
        <scope>...</scope>
</onEvent>
<onEvent partnerLink="pl1" operation="CancelarPeticion" ...>
        <scope>...</scope>
        </onEvent>
</eventHandlers>
</process>
```

EventHandler puede estar asociado a un process completo o a un *scope*.

Correlación de mensajes

Acabamos de ver escenarios en los que un proceso recibe más de un mensaje. Algunos de los actividades asociadas a los mensajes entrantes se pueden utilizar para crear

nuevas instancias de proceso, mientras que otras se utilizan para modelar situaciones en las que una instancia de proceso en ejecución recibe solicitudes adicionales.

Si un mensaje de petición de Servicio Web no da lugar a la creación de una nueva instancia del proceso, ¿cómo "encuentra" la instancia de proceso en ejecución para el cual está diseñado?. BPEL proporciona un mecanismo de correlación llamado *correlationSets*. La mayor observación detrás de este concepto es el hecho de que la mayoría de los mensajes intercambiados entre proceso de negocios y el mundo exterior ya poseen un dato clave que permite identificar una instancia de proceso. BPEL permite definir propieties que representan partes de una información de correlación. La implementación de BPEL está realizada siendo consciente de esas propiedades y su localización en un mensaje usando definiciones de *propety alias*. Finalmente, cada mensaje entrante puede estar asociado con un conjunto de propiedades que juntas correlacionan una petición entrante con una única instancia de proceso.

A continuación un ejemplo. En primer lugar tenemos el WSDL:

```
<wsdl:definitions ...>
...
<wsdl:message name="sendPOResponse">
        <wsdl:part name="confirmation"
element="po:sendPurchaseOrderResponse" />
</wsdl:message>

<wsdl:message name="queryPORequest">
        <wsdl:part name="query"
element="po:queryPurchaseOrderStatus" />
</wsdl:message>
<wsdl:message name="cancelPORequest">
        <wsdl:part name="cancellation"
element="po:cancelPurchaseOrder" />
</wsdl:message>
...
<vprop:property name="customerID" type="xsd:string" />
<vprop:property name="orderNumber" type="xsd:int" />

<vprop:propertyAlias propertyName="tns:customerID"
messageType="tns:sendPOResponse" part="confirmation">
<bpel:query>CID</bpel:query>
</vprop:propertyAlias>

<vprop:propertyAlias propertyName="tns:orderNumber"
messageType="tns:sendPOResponse" part="confirmation">
<bpel:query>Order</bpel:query>
</vprop:propertyAlias>

```

```
<vprop:propertyAlias propertyName="tns:customerID"
messageType="tns:queryPORequest" part="query">
<bpel:query>CID</bpel:query>
</vprop:propertyAlias>

<vprop:propertyAlias propertyName="tns:orderNumber"
messageType="tns:queryPORequest" part="query">
<bpel:query>Order</bpel:query>
</vprop:propertyAlias>

<vprop:propertyAlias propertyName="tns:customerID"
messageType="tns:cancelPORequest" part="cancellation">
<bpel:query>CID</bpel:query>
</vprop:propertyAlias>

<vprop:propertyAlias propertyName="tns:orderNumber"
messageType="tns:cancelPORequest" part="cancellation">
<bpel:query>Order</bpel:query>
</vprop:propertyAlias>
...
</wsdl:definitions>
```

La definición de los mensajes WSDL que llevan datos de correlación han sido mostrados anteriormente. Las dos propiedades (*customerID* y *orderNumber*) son usadas para identificar de manera única a una instancia de proceso. Para cada mensaje WSDL y cada propiedad, las definiciones de *property alias* especifican dónde la propiedad puede estar localizada dentro del mensaje. A continuación se explica de manera más detallada.

Tenemos 2 mensajes de entrada diferentes que serán recibidos en la composición BPEL. Cuando lleguen deberán ir a la instancia correcta del proceso ya que yo no puedo interferir en las transacciones de otra persona ni esa otra persona en las mías.

Por otro lado, la única manera de identificar a la instancia a la que pertenece un mensaje es haciendo uso de *customerID* y *orderNumber*. Así que a cada mensaje se le debe asociar de alguna manera estos dos datos. Si no se les asocian no se podrá identificar la instancia de proceso a la que pertenece y los usuarios pueden interferir entre sí. La única forma de asociarlos es haciendo uso de los *propertyAlias*. Si lo expresamos de manera "matemática" si se pudiera decir así tendríamos:
message que será enviado/recibido + identificador de instancia de proceso = propertyAlias

```
<process name="purchaseOrderProcess" ...>
<correlationSets>
<correlationSet name="PurchaseOrder"
properties="cor:customerID cor:orderNumber" />
</correlationSets>
...
```

```
<eventHandlers>
<onEvent partnerLink="purchasing"
operation="queryPurchaseOrderStatus" ...>
        <correlations>
        <correlation set="PurchaseOrder" initiate="no" />
        </correlations>
                <scope>...</scope>
</onEvent>
<onEvent partnerLink="purchasing"
operation="cancelPurchaseOrder" ...>
        <correlations>
        <correlation set="PurchaseOrder" initiate="no" />
        </correlations>
                <scope>...</scope>
</onEvent>
</eventHandlers>
...
<sequence>
<receive partnerLink="purchasing"
operation="sendPurchaseOrder"
 ... createInstance="yes"/>
...
<reply partnerLink="purchasing"
operation="sendPurchaseOrder" ...>
        <correlations>
        <correlation set="PurchaseOrder" initiate="yes" />
        </correlations>
</reply>
...

...
</sequence>
</process>
```

En el proceso mostrado con anterioridad, la instancia del proceso es creada cuando se recibe una nueva orden de compra. La recepción de la orden de compra es confirmada mediante un *reply*. El mensaje de réplica contiene la información de correlación que debe ser presentada en las subsecuentes peticiones que son dirigidas a esta misma instancia de proceso. Es decir, se les da valor a las properties *CustomerID* y *orderNumber*. Cada manejador de evento por lo tanto referencia al mismo conjunto de correlación.

El conjunto de correlación es iniciado cuando el *reply* de confirmación de la orden de compra es enviado. No se puede modificar después de ese momento ya que identifica la instancia de proceso. Los manejadores de evento para estatus de la orden de compra y cancelación de la misma especifican el atributo *initiate=”no”*. Cuando estas operaciones son invocadas, las propiedades de conjunto de correlación (*customerId* y *orderNumber*)

deben tener exactamente los mismo valores como en la actividad *reply*, en otro caso, la petición no será correlada con la correcta instancia de proceso.

A.3.3. Más procesado en paralelo

Con anterioridad se habló de una variación de la actividad repetitiva *forEach*. Esta variación permite ejecutar cada iteración del bucle paralelamente, es decir, que todas comienzan al mismo tiempo. La principal diferencia con *flow* es que el número de ramas no es conocida en el tiempo del modelado de la composición por eso *forEach* se comporta como un *flow* pero con *N* actividades hijas similares no limitadas por los *links*.

```
<forEach parallel="yes" countername="n">
```

A.3.4. Ejecución retrasada

En algunas situaciones, la ejecución de la lógica de negocio no puede continuar inmediatamente. El proceso tiene que esperar un periodo de tiempo especificado o hasta que un cierto punto en el tiempo es alcanzado. La actividad *wait* indica que el procesamiento será suspendido. Las ramas concurrentes en proceso no se ven afectadas.

```
<wait><!-- Esperar 3 dias y 10 horas-->
<for>'P3DT10H'</for>
</wait>
```

A.3.5. Manipulación de datos simultáneos

Debemos tener cuidado con el acceso a una misma variable global por parte de diferentes actividades al mismo tiempo. Un *scope* de BPEL proporciona una medida para controlar el acceso a datos globales. El atributo *isolated* cuando su valor es *yes* protege el acceso a variables compartidas, no permitiendo que las actividades de un *scope* puedan acceder a ellas mientras las de otro, en el que se utilizan las mismas variables, no hayan terminado.

```
<process ...>
<variables>
<variable name="global" element="..." />
</variables>
<flow>
<scope name="S1" isolated="yes">
```

```
<sequence>
...
<invoke ... outputVariable="global" />
...
</sequence>
</scope>
<scope name="S2" isolated="yes">
<sequence>
...
<assign>
<copy>
<from>...</from>
<to variable="global" />
</copy>
</assign>
...
</sequence>
</scope>
</flow>
</process>
```

A.3.6. Lenguajes de expresión y lenguajes de consulta

Con el propósito de acceder a datos y manipulación de los mismo, muchos elementos en BPEL contienen una expresión o consulta. Por defecto, eso está especificado como un literal de XPath 1.0.

B Operadores de mutación

Éstos se han clasificado en cuatro categorías, de acuerdo con el tipo de elemento sintáctico de WS-BPEL con el que se relacionan.

Las categorías se identifican con una letra mayúscula y son las siguientes:

- mutación de identificadores (I)
- mutación de expresiones (E)
- mutación de actividades (A)
- mutación de condiciones excepcionales y eventos (X)

Dentro de cada categoría se definen varios operadores de mutación que se identifican mediante tres letras mayúsculas: la primera de ellas coincide con la que identifica la categoría a la que pertenece el operador, mientras que las dos últimas identifican al operador dentro de la categoría.
Los operadores definidos modelan fallos que podrían cometerse al generar una composición WSBPEL 2.0, habiéndose tenido en cuenta que normalmente éstas no se suelen escribir de forma directa, sino que se utilizan herramientas gráficas como por ejemplo NetBeans ó Eclipse. Por lo que muchos de los fallos que pueden aparecer en programas escritos en otros lenguajes, debidos a errores al teclear código, no aparecerán en WSBPEL.

B.1. Descripción de los operadores de mutación

B.1.1. Operadores de Mutación de Identificadores

Un error común que puede cometerse al escribir un programa es cambiar el identificador de una variable por el de otra; si ambas son del mismo tipo, esto no provocará un fallo que pueda ser detectado por el compilador. El operador **ISV** modela este tipo de fallos, sustituyendo el identificador de una variable por el de otra del mismo tipo; esta sustitución sólo se realizará si ambas pertenecen al mismo ámbito.

B.1.2. Operadores de Mutación de Expresiones

Se han definido operadores para todos los tipos de expresiones que podemos encontrar en WS-BPEL 2.0, pero sólo se han considerado aquellos operadores de mutación que sustituyen un operador por otro del mismo tipo, ya que creemos que éstos son los fallos que pueden cometerse con mayor frecuencia. Así tenemos:

- **EAA** :Sustituye un operador aritmético (+, -, *, div, mod) por otro del mismo tipo.
- **EEU** :Elimina el operador - unario de cualquier expresión en la que aparezca. No hemos considerado el añadirlo porque creemos que modelaría un fallo poco frecuente.
- **ERR** :Sustituye un operador relacional (<, >, <=, >=, =, !=) por otro del mismo tipo.
- **ELL** :Sustituye un operador lógico (and, or) por otro del mismo tipo.
- **ECC** :Sustituye un operador de camino (/, //) por otro del mismo tipo.
- **ECN** :Tiene como dominio las constantes numéricas que aparecen en el programa. Modela los errores que pueden cometerse al introducir constantes numéricas, modificándolas de varias formas: incrementa o decrementa su valor en una unidad, añade o elimina alguna cifra.
- **EMD**: Modifica una expresión de duración de dos formas: la sustituye por 0, lo que implica que la condición se cumple inmediatamente, y por la mitad del valor inicial; esto permite verificar si el margen de seguridad especificado por la duración es adecuado. Este operador es aplicable a la actividad *wait*, y al elemento onAlarm de la actividad *pick* y de los manejadores de eventos.
- **EMF**: Similar al anterior. Modifica una expresión de fecha límite.

B.1.3. Operadores de Mutación de Actividades

Algunos de los operadores de esta categoría modelan el fallo que puede cometerse al elegir una actividad que no es la más adecuada para las acciones que se deben realizar, sustituyendo una actividad por otra. Los demás modelan la elección de un valor incorrecto para los atributos de las actividades, sustituyendo para ello su valor actual por otro valor válido. Estos operadores se han clasificado en dos tipos, los relacionados con la concurrencia y los no concurrentes.

B.1.4. Relacionados con la concurrencia.

Las actividades *receive* y *pick* permiten recibir mensajes. Ambas pueden especificar un atributo denominado *createInstance* que, si está activado, creará una nueva instancia del proceso cuando llegue un nuevo mensaje y, si no lo está, hará que el nuevo mensaje

sea consumido por una de las instancias que ya existen. El operador **ACI** cambia el atributo createInstance de "yes" a "no" sólo en el caso de que el proceso tenga más de una actividad con dicho atributo a "yes", ya que en caso contrario el mutante no sería válido.

La actividad *forEach* permite ejecutar un conjunto de actividades un número fijo de veces. Existen dos variantes de *forEach* que se diferencian en el modo de ejecutar las iteraciones, secuencial o paralelamente. El operador **AFP** cambia el carácter secuencial de una actividad forEach por paralelo, modificando el valor del atributo parallel de "no" a "yes"

La actividad sequence ejecuta secuencialmente las actividades que contiene, mientras que *flow* las ejecuta en paralelo. El operador **ASF** cambia una actividad sequence por *flow*; el cambio contrario no se realiza porque dicha mutación generaría un mutante equivalente.

WS-BPEL proporciona un mecanismo para proteger el acceso concurrente a datos globales, se trata del atributo *isolated* de un *scope*. Cuando su valor es "yes" se protege el acceso a variables compartidas, no permitiendo que las actividades de un scope puedan acceder a ellas mientras las de otro, en el que se utilizan las mismas variables, no hayan terminado. El operador **AIS** pone el atributo *isolated* de un scope en el que se manipulan variables compartidas a "no".

B.1.5. No Concurrentes

La actividad *if* permite definir las actividades a ejecutar en función del valor verdadero o falso de un conjunto de condiciones. Permite especificar opcionalmente uno o varios elementos *elseif* y un elemento *else*. El operador **AIE** elimina un elemento *elseif* o el elemento *else* de una actividad *if*, modelando el olvido de una rama de esta actividad.

Aparte de *forEach*, WS-BPEL proporciona otras dos actividades repetitivas, *repeatUntil* y *while*. El operador **AWR** cambia una actividad while por otra *repeatUntil* y viceversa, modelando el error cometido al no elegir el tipo de actividad repetitiva adecuada.

A todas las actividades de WS-BPEL pueden asociarse los contenedores *sources* y *targets*, los cuales contienen elementos *source* y *target*, respectivamente. Estos elementos permiten sincronizar actividades. Se puede especificar para el contenedor *targets* un elemento denominado *joinCondition*, cuyo valor es una expresión *booleana*. Cuando no se especifica este elemento, la condición que se considera es la disyunción del estado de todos los target de esta actividad. El operador **AJC** elimina el elemento joinCondition.

El operador **ASI** intercambia el orden de dos actividades dentro de una actividad sequence. Si no existe dependencia de datos entre ellas se producirán mutantes equivalentes.

La actividad de recepción de mensajes *pick* puede especificar mediante los elementos *onMessage* las actividades a realizar cuando el proceso recibe un mensaje determinado. El elemento *onAlarm* indica las acciones a ejecutar si no se recibe ningún mensaje en un tiempo determinado. El operador **APM** elimina un elemento onMessage, siempre que haya más de uno, modelando el error que se cometería al olvidar la posible recepción de un mensaje. El operador **APA** elimina el elemento onAlarm, modelando el error que se produciría al olvidarlo. También se puede aplicar al elemento onAlarm de un manejador de eventos.

B.1.6. Operadores de Mutación Relacionados con las Condiciones Excepcionales y Eventos

Los manejadores de fallos permiten especificar mediante los elementos *catch* las actividades a ejecutar en caso de que se produzca un fallo determinado, y mediante el elemento *catchall* las relacionadas con cualquier otro fallo no especificado anteriormente. El operador **XMF** elimina un elemento *catch* o el elemento *catchall* de un manejador de fallos; modelando el olvido de incluir un manejador específico para un fallo determinado, o bien el de un manejador por omisión.

La actividad *reply* permite enviar mensajes de respuesta, o bien mensajes de fallo, que se especifican mediante el atributo *faultname*. El operador **XRF** elimina este atributo.

A veces, si se produce un fallo durante la ejecución de un proceso, es necesario deshacer el trabajo que ya ha sido hecho; para ello se dispone del manejador de compensación. Por otro lado, dentro de un *scope* se puede definir un manejador de terminación, que indica las actividades a realizar si se debe terminar la ejecución de dicho *scope*. El operador **XMC** elimina la definición de un manejador de compensación, y el operador **XMT**, la de un manejador de terminación. En ambos casos, WSBPEL los sustituirá por el manejador por omisión de cada uno de ellos.

La actividad *throw* permite lanzar un fallo determinado, cuyo nombre se especifica mediante el atributo *faultName*. El operador **XTF** cambia el nombre del fallo lanzado por una actividad *throw* por otro del mismo ámbito, modelando la confusión en la especificación del fallo a lanzar. La actividad *rethrow* permite volver a lanzar un fallo previamente capturado. El operador **XER** elimina una actividad *rethrow*, modelando el

olvido de su inclusión en el manejador de fallos.

Los manejadores de eventos pueden contener elementos *onEvent*, que determinan las acciones a realizar cuando se produce un evento dado, y un elemento *onAlarm*. El operador **XEE** elimina un elemento *onEvent* de un manejador de eventos, modelando el olvido a la hora de especificar un evento que puede recibir el proceso.

C Algunos errores al programar con WS-PBEL en GAmera

C.1. Error desplegando la composición

- *Error: Could not find partner or client with the name PL_AVAL.*
  No existe concordancia entre el nombre de algún partnerLink en el fichero *.bpel* y el nombre de algún tes:partner del fichero *.bpts*
- *Error: An element 'copy' from the BPEL namespace was added as an extensibility element since it appeared in an invalid location within the process.: in process.pdd .*
  Un elemento aparece en una localización dentro del bpel que no le corresponde. En este caso es una actividad <copy>. Copy debe estar dentro de una actividad <assign>pero en este caso la hemos sacado fuera para observar el error.
- *Exception in thread "client"java.lang.IllegalArgumentException: Invalid uri 'http://localhost:$HttpDefaultPort/BPELSampleSynchronousWSDLService/ BPELSampleSynchronousWSDLPort': invalid port number*
  at org.apache.commons.httpclient.HttpMethodBase.<init> (HttpMethodBase.java:219)
  at org.apache.commons.httpclient.methods.ExpectContinueMethod.<init> (ExpectContinueMethod.java:92) ....
  El problema está en el valor del atributo "location" que tiene el carácter '/' al final:

```
1 <port name="SynchronousSamplePort" binding="tns:
     SynchronousSampleBinding">
2 <soap:address location="http://localhost:8080/active-bpel/
     services/SynchronousSampleService/>
3 </port>}
```

La solución simplemente consiste en quitar la barra del final:

```
1 <port name="SynchronousSamplePort" binding="tns:
     SynchronousSampleBinding">
2 <soap:address location="http://localhost:8080/active-bpel/
     services/SynchronousSampleService"/>
```

```
3  </port> }
```

C.2. Error en el envío de mensajes

- *Timeout while waiting for incoming synchronous message.*
  Se agotó el tiempo de espera en la entrada de un mansaje en el fichero *.bpts*
  Este error se puede producir por múltiples razones.
  El fichero *.bpts* espera la entrada de un mensaje proveniente desde *.bpel* pero éste no llega. Esta situación se da normalmente cuando durante la ejecución del *.bpel* se produce algún tipo de error inesperado y no se completa la ejecución normal de la composición por tanto, no se envían todos los mensajes que debían haber sido enviados y la ejecución falla.
- *HTTP Error while sending out synchronous message: Read timed out*
  Error al enviar un mensaje desde la composición bpel hasta el fichero *.bpts*. Como ocurre en el caso anterior, las razones para este error pueden ser varias. Una de ellas puede ser que el mensaje que se espera como entrada en el *.bpts* debe tener 2 partes al menos, y tan solo tiene 1.

C.3. Errores en el *.bpts*

- *Error: BPTS is invalid: .../LoanApprovalExtended.bpts:0: error: cvc-complex-type.2.4c: Expected elements :send@http://www.bpelunit.org/schema/testSuite' before the end of the content in element receiveSend@http://www.bpelunit.org/schema/testSuite*

  En esta ocasión el error se produce porque algún elemento de envío de mensajes de BpelUnit no está construido correctamente. En este ejemplo tenemos que un elemento <sendReceive>carece del elemento hijo <send>. El error podría haber sido producido por send, receive, data ...
- *Error: Specified service "{http://j2ee.netbeans.org/wsdl/morosidadASNEF_WSDL} morosidadASNEF_WSDLServices"was not found in partner WSDL {http://j2ee.netbeans.org/wsdl/morosidadASNEF_WSDL} morosidadASNEF_WSDL*

  Tanto este error como similares son debidos a que cuando especificamos en algún elemento para recibir o enviar mensajes los atributos que damos son incorrectos. Este error se solventa escribiendo el nombre correcto del servicio: morosidadASNEF_WSDLService

- *[Fatal Error] BPELSampleSynchronous.bpts:47:6: The element type "tes:receiveSend"must be terminated by the matching end-tag "</tes:receiveSend>". SAX parsing error: The element type "tes:receiveSend"must be terminated by the matching end-tag "</tes:receiveSend>".* No se ha cerrado convenientemente la etiqueta de alguna actividad.

- *Error: Cannot read WSDL file for partner BPELSample: File "CLIENTE.wsdl"not found.* No existe un fichero .wsdl con un nombre determinado.
- *Error: Specified operation "BPELSampleSynchronousWSDLPort"was not found in binding "{http://j2ee.netbeans.org/wsdl/BPELSampleSynchronous/BPELSampleSynchronousWSDL}BPELSampleSynchronousWSDLBinding"in partner WSDL "{http://j2ee.netbeans.org/wsdl/BPELSampleSynchronous/BPELSampleSynchronousWSDL}BPELSampleSynchronousWSDL"* La operación, binding o servicio especificado no existe.
- *Condition 'dictamen='true" did not hold: Condition failed. Obtained value was 'false', expected 'true'*
  La anterior situación no es un error de ejecución propiamente dicho. Se da cuando la ejecución de la composición ha sido correcta y el intercambio de mensajes también pero los resultados obtenidos no son los que se esperaban.

D De NetBeans a GAmera

D.1. Transformación de una composición ejecutable en NetBeans a ejecutable en GAmera

A continuación se describirá el proceso de "transformación" de una composición creada y ejecutable en NetBeans a composición ejecutable en GAmera.

D.1.1. Creación de un proyecto en NetBeans

En la parte superior izquierda pinchamos en *File* y seguimos la siguiente secuencia.

1. *File*.
2. *New Project*.
3. SOA.
4. BPEL *Module*.

Damos un nombre a la composición, por ejemplo *BpelSampleSynchronous* como se muestra en la figura D.1

A continuación pinchamos con el botón derecho sobre *Process Files* y seguimos la secuencia:

1. *New* WSDL *File*.
2. Elegir el tipo de documento WSDL "*Concret*".
3. Lo demás lo dejamos por defecto.

Arrastramos el fichero *.wsdl* al gráfico representativo del *.bpel*. De este modo crearemos un *partnerLink* hacia el cliente como muestra la figura D.2

Colocamos una actividad *receive* y un *reply* en el cuerpo del bpel con ayuda de la "paleta" que esta en la parte diestra del entorno.

Debemos conectar el *partnerLink* del cliente con el *receive* de la composición. Para ello, sobre la actividad *receiv*e en el icono de editar, rellenaremos con las opciones por

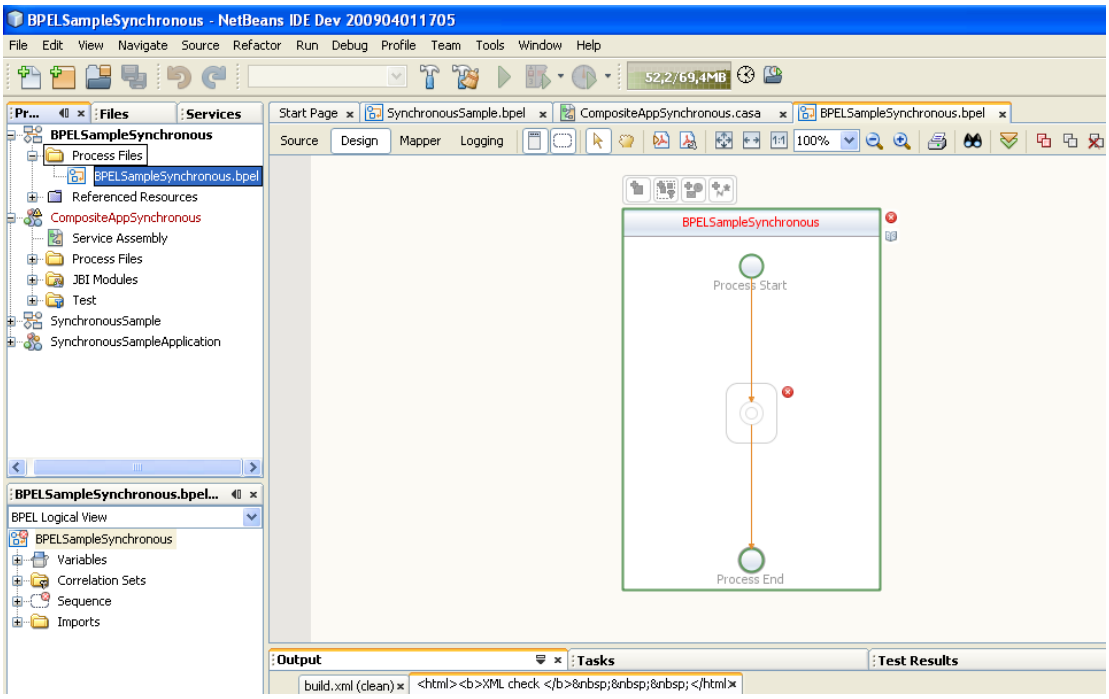

Figura D.1: Creando la composición en NetBeans

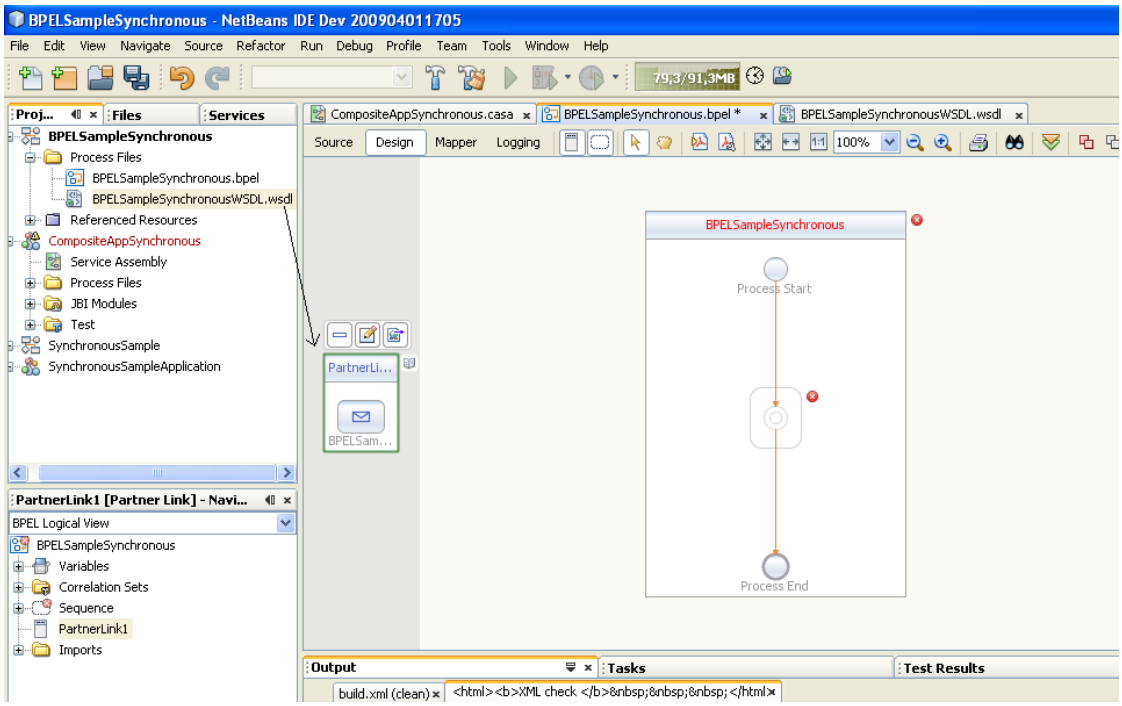

Figura D.2: Estableciendo PartnerLink

defecto que nos aparecerán en las ventanas desplegables. Deberemos darle a la opción *create* para crear una variable de entrada en el *.bpel* ya que hasta el momento no tenemos ninguna variable. Ésta también se rellenará por defecto.
Lo mismo haremos con la réplica. Pulsando sobre el icono de editar del *reply* rellenaremos con los valores por defecto. La variable de devolución deberá ser también creada y ésta será de tipo *normal* (la otra opción es que la variable sea de un tipo para devolver un falta (error)).

Hasta ahora se recibe una petición (dato de entrada) y se devuelve una respuesta. Quedan todavía por hacer 2 cosas importantes:

- Trabajar con esa petición que recibimos.
- Interactuar con un "socio".

Para ello crearemos un nuevo *.wsdl* y posteriormente un nuevo *.bpel*.
Sobre *Process Files*, botón derecho y creamos un*.wsdl*.
Repetimos el proceso anterior arrastrando el *.wsdl* a la representación gráfica del *.bpel* pero esta vez al lado derecho de la composición ya que en este caso el *.wsdl* ejercerá como socio y no como cliente.

Deberemos crear un enlace entre la composición y el nuevo *.wsdl*. Para ello con ayuda de la paleta, arrastramos la actividad *invoke* hasta el cuerpo del *.bpel*, entre las actividades *receive* y *reply*. De forma análoga al *reply* y *receive* unimos ambos editando su contenido.

Ahora debemos hacer uso de la entrada del cliente y proporcionarle una salida al mismo. Para ello copiaremos la variable de entrada de la composición bpel *BPELSampleSynchronousWSDLOperationIn* con la variable de entrada al socio *PartnerWSDLOperationIn*. De igual modo copiaremos lo que nos devuelva el socio del trabajo *BPELSampleSynchronousWSDLOperationOut* a la salida final hacia el cliente *BPELSampleSynchronousWSDLOperationOut*. Observe la figura D.3

Esto lo haremos a través de actividades *assign*. En la anterior imagen podemos ver cómo se realiza esta acción. Debemos enlazar la parte *part1* de la entrada del *.bpel* con la parte *part1* de la entrada del socio. De forma similar lo haremos con la salida.

Hasta ahora estamos en la posición de quien recibe una petición, intercambia mensajes con un socio y devuelve una salida al cliente como indica la figura D.4

Si queremos çerrar el círculo” queda por hacer el contenido del socio. Esto se hace con el nuevo *.bpel* que hablábamos antes el cual recibirá una petición y devolverá algo. Lo mas sencillo que podemos hacer en el cuerpo del nuevo *.bpel* es que devuelva exactamente lo que recibe.

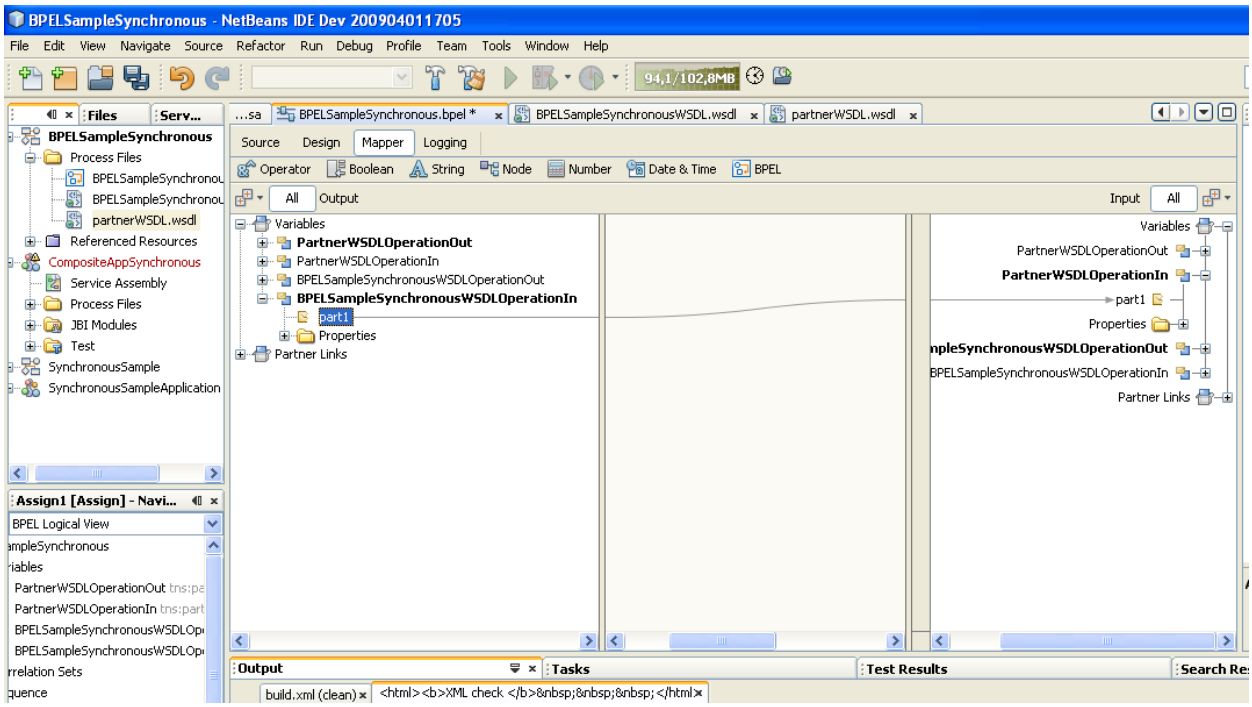

Figura D.3: Copiando partes de variables

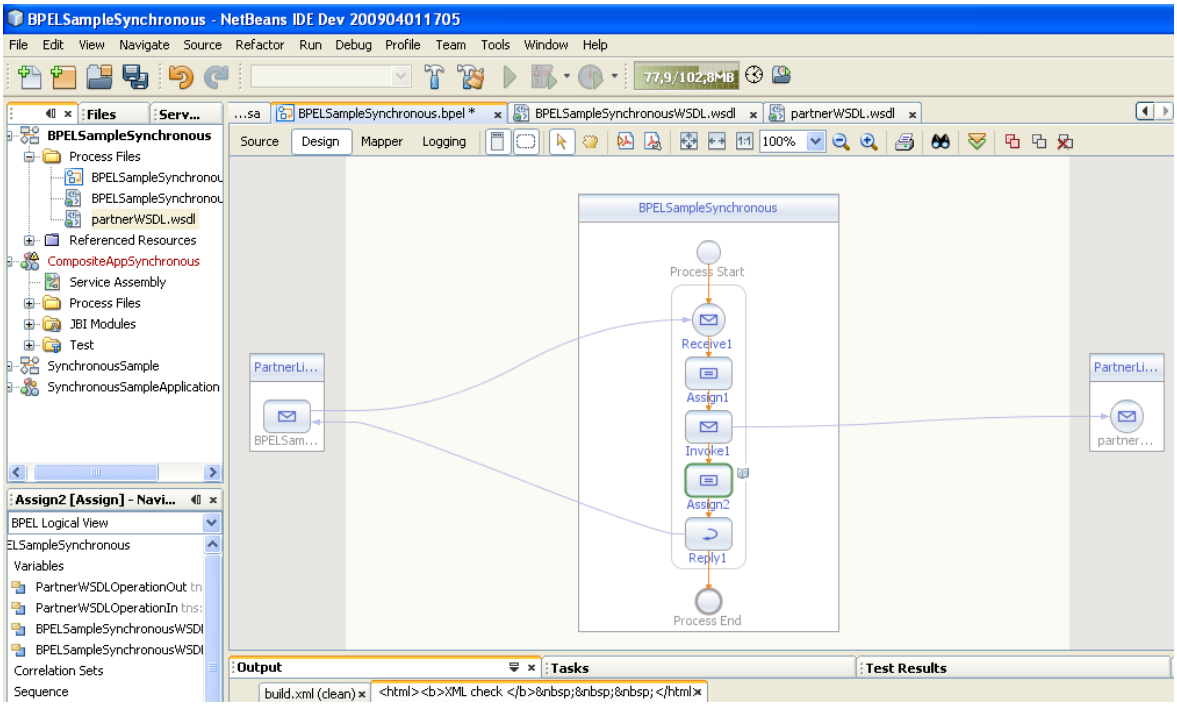

Figura D.4: Estructura de la composición

Una vez hecho todo esto comprobamos que se ha creado toda la composición correctamente, para ello pulsamos con el botón derecho sobre el menú de la izquierda y elegimos la opción *build*.

D.1.2. Creación de la aplicación

Ahora entramos en la segunda fase de la creación de la composición. En esta crearemos la aplicación que nos permitirá la ejecución de la composición. Para ello:

1. *File*
2. *New Project*
3. SOA
4. *Composite Application*
5. Establecer un nombre: *compositeAppSynchronoues*

Debemos ligar este *composite application* con nuestra composición. Lo haremos de la siguiente forma:

- Pinchamos con el Botón derecho sobre *compositeAppSynchronoues*
- *Add* JBI *module*
- Pinchamos sobre *BpelSampleSynchronous*. A la derecha en un recuadro nos aparece un fichero *.jar*.
- Seleccionamos el fichero *jar*
- *Add project jar files*

Ahora comprobamos que lo que llevamos hecho hasta ahora esta bien. Esto lo conseguimos pulsando con el botón derecho sobre el Application y seleccionando *deploy*.

Nos aparecerá el fichero *build.xml* en el que se debe encontrar mensajes similares a los siguientes:

```
Building jar: D:.../BPELSampleSynchronouesApp.zip
run-jbi-deploy:
[deploy-service-assembly]
    Deploying a service assembly...
        host=localhost
        port=4848
        file=D:.../BPELSampleSynchronouesApp.zip
[start-service-assembly]
    Starting a service assembly...
        host=localhost
        port=4848
        name=BPELSampleSynchronouesApp
run:
```

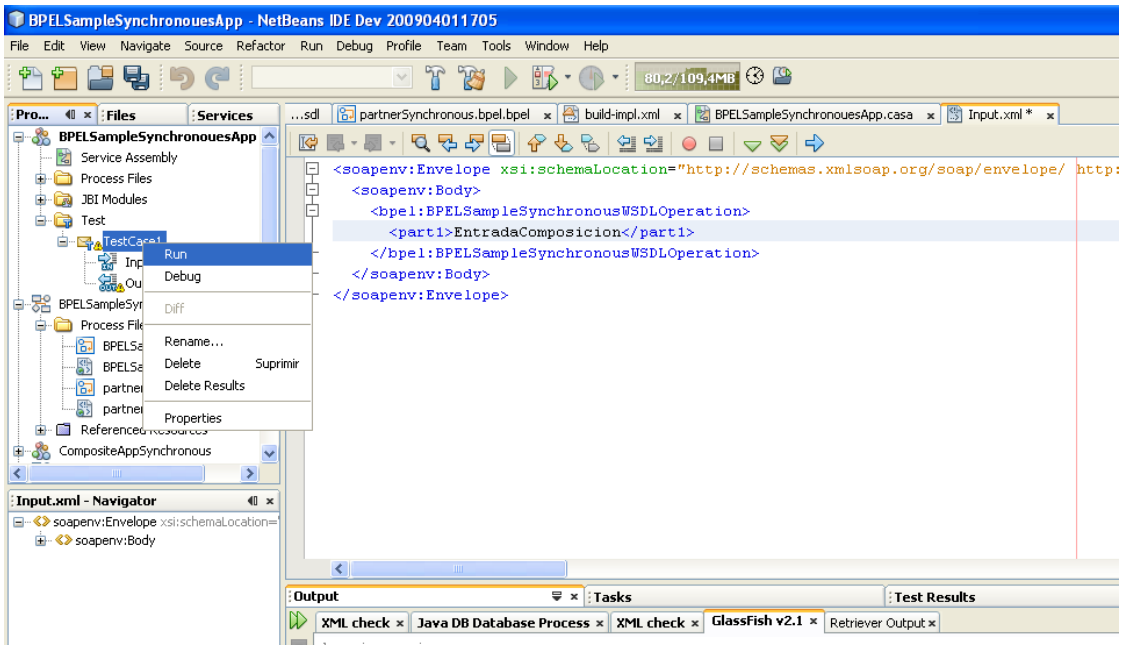

Figura D.5: Ejecutando la composición

```
14 BUILD SUCCESSFUL (total time: 2 seconds)
```

Hasta ahora sabemos que nuestra composición se despliega correctamente. Ahora debemos ejecutarla. Sobre la carpeta *Test* botón derecho:

1. *New test case*
2. Damos un nombre
3. Seleccionamos el *.wsdl* correspondiente al cliente (ya que esa va a ser la entrada a la composición)
4. Seleccionamos la operación de entrada.

Nos aparecerá un fichero *input.xml* en el que deberemos introducir la cadena de texto (en este caso) que recibirá como entrada la composición.
Ahora ejecutamos el caso de prueba 1. Podemos verlo en la figura D.4

No aparecerá unos mensajes de error (esto ocurre siempre en la primera ejecución de cada caso de prueba). No obstante la salida es la esperada como se puede apreciar en la siguiente imagen. La variable de respuesta tiene el valor que le dimos en la entrada.

D.1.3. De NetBeans a GAmera

- Partimos del directorio del proyecto de NetBeans.

En él encontramos los siguientes subdirectorios y ficheros:

- *build*
- *nbproject*
- *src*
- *build.xml*
- *catalog.xml*

  Tan sólo *src* es la que nos interesa, todo lo demás lo podemos descartar.

- Dentro de *src* el fichero *.bpel* que contiene el comportamiento del socio también lo vamos a eliminar.

Ya que esto lo vamos realizar en GAmera usando un BPELUnit que permite usar *mockups* que sustituyen a los servicios webs reales.

- Creamos un fichero *.bpts* que será el que actúe de mockup.

El contenido se expondrá a continuación al mismo tiempo que se detallan algunas de sus partes más importantes:

```
<?xml version="1.0" encoding="UTF-8"?>

<tes:testSuite
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://j2ee.netbeans.org/wsdl/
BPELSampleSynchronous/BPELSampleSynchronousWSDL"
xmlns:ns3="http://j2ee.netbeans.org/wsdl/
BPELSampleSynchronous/partnerWSDL"
xmlns:tes="http://www.bpelunit.org/schema/testSuite"
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
executable"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/
business-process/" >
```

Damos un nombre:

```
<tes:name>BPELSample</tes:name>
```

El siguiente valor es fijo:

```
<tes:baseURL>http://localhost:7777/ws</tes:baseURL>
```

Y a continuación:

```
<tes:deployment>
  <tes:put name="BPELSample" type="activebpel">
  <tes:wsdl>BPELSampleSynchronousWSDL.wsdl</tes:wsdl>
  <tes:property name="BPRFile">BPELSample.bpr</tes:property>
  </tes:put>
  <tes:partner name="PartnerLink2" wsdl="partnerWSDL.wsdl"/>
</tes:deployment>
```

Creamos un caso de prueba:

```
<tes:testCases>
<tes:testCase name="testCase1" basedOn="" abstract="false"
vary="false">
```

Simulamos el intercambio de mensajes con el cliente:

```
<tes:clientTrack>
```

Los mensajes se intercambian de forma síncrona. Esto en el mockup se modela utilizando actividades sendReceive.Debemos especificar el servicio, el puerto y la operación.

```
        <tes:sendReceive service="ns1:
           BPELSampleSynchronousWSDLService"
port="BPELSampleSynchronousWSDLPort"
operation="BPELSampleSynchronousWSDLOperation">
```

Mensaje que va a enviar el cliente:

```
                <tes:send fault="false" >
                <tes:data>
                        <ns1:part1>'31709567L'</ns1:part1>
                </tes:data>
                </tes:send>
```

Mensaje que se va a devolver al cliente. Se puede especificar una condición de devolución que debe cumplirse para saber que se ha ejecutado correctamente la composición.:

```
                <tes:receive fault="false">
                <tes:condition>
                <tes:expression>part1</tes:expression>
                <tes:value>"'false'"</tes:value>
```

```
                </tes:condition>
                </tes:receive>
        </tes:sendReceive>
</tes:clientTrack>
```

Simulamos el intercambio de mensajes con el socio:

```
<tes:partnerTrack name="PartnerLink2">
        <tes:receiveSend service="ns3:partnerWSDLService"
port="partnerWSDLPort2" operation="partnerWSDLOperation">
                <tes:send fault="false">
                <tes:data>
                        <part1>'false'</part1>
                </tes:data>
                </tes:send>
                <tes:receive fault="false"/>
                </tes:receiveSend>
</tes:partnerTrack>
                </tes:testCase>
        </tes:testCases>
</tes:testSuite>
```

- En el fichero BPELSampleSynchronousWSDL.wsdl debemos modificar esto:

```
<soap:address location="http://localhost:HttpDefaultPort}/
BPELSampleSynchronousWSDLService/BPELSampleSynchronousWSDLPort"/>
```

Por esto otro:

```
<soap:address location="http://localhost:8080/active-bpel/
services/BPELSampleSynchronousWSDLService"/>
```

Bibliografía

[1] Keith Checkley. *Manual para el análisis del riesgo de crédito*. GESTION 2000.com, 2003.

[2] Ministerio de Administraciones Públicas. Metrica V3. Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. http://www.csi.map.es/csi/metrica3/index.html.

[3] Active Endpoints. ActiveBPEL Engine. www.activevos.com/community-open-source.php.

[4] Euroresidentes. Conceptos de préstamos personales-Euroresidentes. http://www.euroresidentes.com/vivienda/hipotecas/diccionario/aval.htm.

[5] Universität Hannover. BPELUnit. http://www.se.uni-hannover.de/forschung/soa/bpelunit/.

[6] Domínguez Jiménez, Juan José. Medina Bulo, Inmaculada. Estero Botaro, Antonia. Jisweb08. *Operadores de mutación para WS-BPEL 2.0*, 2008.

[7] Domínguez Jiménez, Juan José. Medina Bulo, Inmaculada. Estero Botaro, Antonia. *Una arquitectura para la generación de casos de prueba de composiciones WS-BPEL basada en mutaciones*, 2009.

[8] Domínguez Jiménez, Juan José. Medina Bulo, Inmaculada. Estero Botaro, Antonia. Jisweb09. *Un sistema para la generación automática de mutantes de composiciones WS-BPEL*, 2009.

[9] Prestamos-Personales.info. Glosario de conceptos sobre préstamos e hipotecas. http://www.prestamos-personales.info/index.php?module=de&tid=10.

[10] todoprestamos.es. Conceptos préstamos personales. http://www.todoprestamos.es/creditos-personales/.

[11] W3C. Especificación WSBPEL 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[12] W3C. Especificación WSDL. http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/.

[13] W3C. Especificación XML Schema. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[14] W3C. Especificación XPath. http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[15] W3C. Servicios Web-W3C . http://www.w3c.es/divulgacion/guiasbreves/ServiciosWeb.

[16] W3C. WSDL-W3C. http://www.w3.org/TR/wsdl.

[17] Wikipedia. LaTex. http://es.wikipedia.org/wiki/LaTeX.

[18] Wikipedia. WSDL. http://es.wikipedia.org/wiki/WSDL.

[19] Wikipedia. XML-Schema. http://es.wikipedia.org/wiki/XML_Schema.

[20] Wikipedia. XPath. http://es.wikipedia.org/wiki/XPath.